Computer Standards & Interfaces 84 (2023) 103676

Contents lists available at ScienceDirect

Computer Standards & Interfaces

journal homepage: www.elsevier.com/locate/csi

EDALoCo: Enhancing the accessibility of blockchains through a low-code approach to the development of event-driven applications for smart contract management

Jesús Rosa-Bilbao [a], Juan Boubeta-Puig [a,*], Adrian Rutle [b]

[a] UCASE Software Engineering Research Group, Department of Computer Science and Engineering, University of Cadiz, Avda. de la Universidad de Cádiz 10, 11519 Puerto Real, Cádiz, Spain

[b] Department of Computer Science, Electrical Engineering and Mathematical Sciences, Western Norway University of Applied Sciences, Bergen, Norway

ARTICLE INFO

Keywords:
Low-code
Smart contract
Blockchain
Event-driven application

ABSTRACT

Blockchain is a cutting-edge technology based on a distributed, secure and immutable ledger that facilitates the registration of transactions and the traceability of tangible and intangible assets without requiring central governance. The agreements between the nodes participating in a blockchain network are defined through smart contracts. However, the compilation, deployment, interaction and monitoring of these smart contracts is a barrier compromising the accessibility of blockchains by non-expert developers. To address this challenge, in this paper, we propose a low-code approach, called EDALoCo, that facilitates the development of event-driven applications for smart contract management. These applications make blockchain more accessible for software developers who are non-experts in this technology as these can be modeled through graphical flows, which specify the communications between data producers, data processors and data consumers. Specifically, we have enhanced the open-source Node-RED low-code platform with blockchain technology, giving support for the creation of user-friendly and lightweight event-driven applications that can compile and deploy smart contracts in a particular blockchain. Additionally, this platform extension allows users to interact with and monitor the smart contracts already deployed in a blockchain network, hiding the implementation details from non-experts in blockchain. This approach was successfully applied to a case study of COVID-19 vaccines to monitor and obtain the temperatures to which these vaccines are continuously exposed, to process them and then to store them in a blockchain network with the aim of making them immutable and traceable to any user. As a conclusion, our approach enables the integration of blockchain with the low-code paradigm, simplifying the development of lightweight event-driven applications for smart contract management. The approach comprises a novel open-source solution that makes data security, immutability and traceability more accessible to software developers who are non-blockchain experts.

1. Introduction

Blockchain [1] is an emergent technology that is based on a distributed ledger where the blocks are linked and encrypted to protect the security and privacy of transactions. Some studies estimate that, within a few years, this technology will be used worldwide, reaching a volume of approximately 2 trillion per year of goods and services [2]. As an example, it is having a considerable impact on areas such as e-health [3–5], smart industry [6], cybersecurity [7,8], smart cities [9], education [10] and voting [11].

Blockchain emerges as a solution to address the challenge of ensuring the security, integrity, traceability, immutability and transparency of data generated by devices [12] and services thanks to its decentralized nature. One of the main advantages of the blockchain technology is that it does not require trusted third parties or a centralized certification authority for the verification of transactions [13].

All transactions that take place in a blockchain network are grouped into blocks, and each block is cryptographically linked to the previous block upon which it is validated [14]. When a new block is mined, it is replicated across all participating nodes belonging to the network. Specifically, the behavior of blockchain networks can be programmed through the use of smart contracts. These contracts can be used to specify agreements between two or more different parties at design time and whose conditions will be validated at runtime [15].

* Corresponding author.
E-mail addresses: jesus.rosabilbao@alum.uca.es (J. Rosa-Bilbao), juan.boubeta@uca.es (J. Boubeta-Puig), adrian.rutle@hvl.no (A. Rutle).

https://doi.org/10.1016/j.csi.2022.103676
Received 4 May 2022; Received in revised form 13 July 2022; Accepted 31 July 2022
Available online 4 August 2022

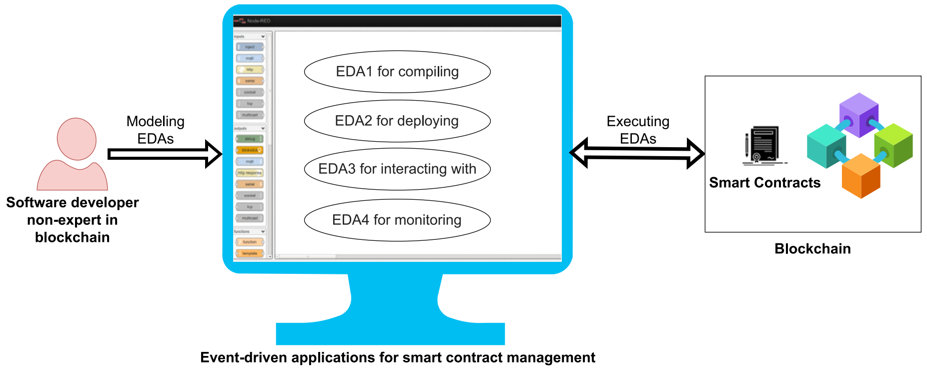

Modeling EDAs
Software developer non-expert in blockchain
EDA1 for compiling
EDA2 for deploying
EDA3 for interacting with
EDA4 for monitoring
Executing EDAs
Smart Contracts
Blockchain
Event-driven applications for smart contract management

Fig. 1. General overview of the EDALoCo approach.

Smart contracts are becoming increasingly common in the business world, reinforced by the digitalization trends in all aspects of the society [16]. Smart contracts are viable because they enable the exchange of digital assets as a means of payment and the execution of code through distributed applications.

However, smart contracts could have bugs or vulnerabilities that result in security breaches and can lead to large economic losses. These bugs and vulnerabilities can be mitigated with the use of good programming practices. To this end, several works have proposed how to develop robust smart contracts [17,18].

Moreover, the accessibility of blockchain technology is currently a handicap for domain experts and even for software developers, since it takes time and requires technical knowledge and effort to develop and compile secure smart contracts, deploy them in a blockchain network, interact with them and monitor them in real time.

To deal with these challenges, this paper aims to propose a *low-code* approach, called EDALoCo, that allows software developers, who are non-blockchain experts, to develop event-driven applications for easily compiling, deploying, monitoring, and interacting with smart contracts in real time (see Fig. 1). This way, software developers do not need to know how blockchain technology works or how to compile, deploy and interact with smart contracts within the network. Thanks to the use of the low-code paradigm [19], the development of event-driven applications allows us to save resources, to improve the quality code, to be more flexible as well as to automate and integrate these applications with other systems.

In order to enhance the accessibility of the blockchain technology, we have extended the Node-RED open-source low-code platform [20] through the development of new nodes that allow the integration of blockchain technology with the low-code paradigm. As a blockchain network, we have chosen Ethereum since it is one of the most widely used and well-known networks of smart contracts, as demonstrated in recent literature [21]. Note that other blockchain networks could be used without requiring implementation changes in these new nodes. This extension will allow us to support the development of event-driven applications capable of managing smart contracts in real time. These applications can be modeled through graphical flows, which specify the communications between data producers, data processors and data consumers [22]. By using software containers, our proposed solution can be run on most lightweight devices, such as Raspberry [23], without worrying about particular configurations or operating systems [24].

Although the approach presented in this paper is generic and can be applied to other domains, we have demonstrated its feasibility by applying it to a specific case study. This case study consists of effective monitoring and obtaining real-time temperatures of COVID-19 vaccines while processing and storing these temperature values within a blockchain network. These recorded temperatures will be immutable and traceable for any user. Thanks to the use of the low-code paradigm, the development of event-driven applications becomes easier [25]. In addition, the case study allows us to also evaluate the effectiveness of our low-code approach to enhance the accessibility of the blockchain network through hiding the implementation details from domain experts and software developers.

The contributions of this paper, which have not been addressed altogether by prior research, are be summarized as follows:

- **C1**: Integrating blockchain and low-code paradigms.
- **C2**: Developing event-driven applications for smart contract management.
- **C3**: Deploying the event-driven applications in lightweight devices.
- **C4**: Proposing an open-source solution.

The remainder of the paper is organized as follows. Section 2 introduces the technologies and paradigms used in this work. Section 3 describes related work. Section 4 presents our low-code approach for smart contract management in a graphical way. Section 5 presents a real-world case study where our proposal has been applied, and Section 6 discusses the obtained results. Finally, Section 7 draws conclusions and outlines future work.

2. Background

In this section, the background on the technologies and paradigms used in our proposal is described: blockchain and low-code.

2.1. Blockchain

Blockchain can be defined as a distributed database in which transactions are recorded and confirmed. All transactions must be verified, recorded and combined with other transactions to create new blocks. These blocks are replicated throughout the network by all nodes participating in the network, thus creating a distributed network (see Fig. 2).

Blockchain networks have a series of stages necessary for a new block to be created and secured. These stages are responsible for collecting and computing the data in blocks, unifying them securely, validating them and maintaining consensus among all the nodes belonging to the network [26].

When a new transaction is recorded in the blockchain network, this information is shared by all users belonging to the network. All blocks are formed from transactions and are identified by a timestamp, which allows to organize them in a sequential order to avoid errors or duplications. All blocks that are recorded in the blockchain network are immutable. Therefore, any participant who wants to check the history of the blockchain network will get the same result and in the same order [27].

Within blockchain networks, several elements can be found. One of the most important elements is the hash function [27]. This function is

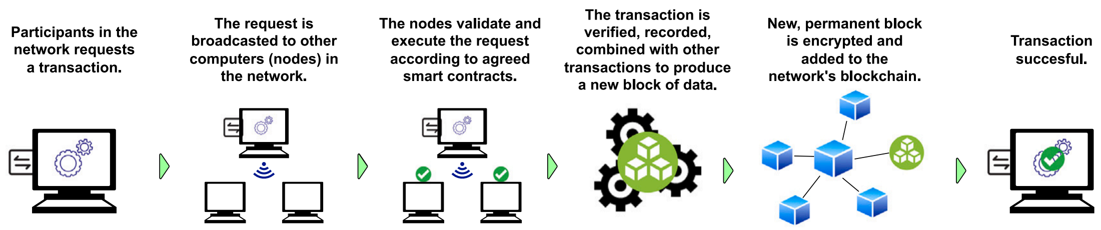

Participants in the network requests a transaction.
The request is broadcasted to other computers (nodes) in the network.
The nodes validate and execute the request according to agreed smart contracts.
The transaction is verified, recorded, combined with other transactions to produce a new block of data.
New, permanent block is encrypted and added to the network's blockchain.
Transaction succesful.

Fig. 2. Operation of transactions in a blockchain network.

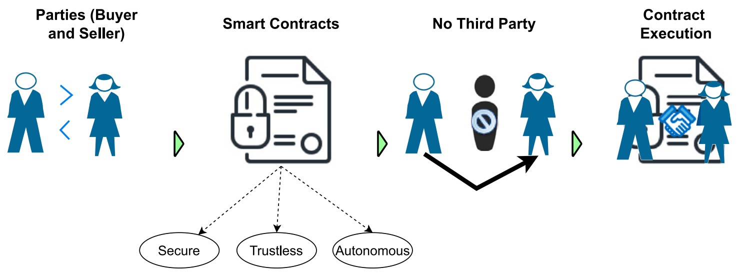


Fig. 3. Operation of smart contracts in a blockchain network.

a cryptographic algorithm that securely links all the blocks in the network, making it impossible to break. The information in each block is used to create a unique sequence of characters that uniquely identifies any block.

Each of the blocks that form a blockchain network has a hash which is added to the next block's data. That is, when a new block is to be mined in the network, it contains information from the immediately preceding block. This linkage allows that if any person tries to manipulate a block, it will cause the whole chain to change. Furthermore, this process is replicated in each block that belongs to a blockchain network. Therefore, if someone tries to manipulate any part of a network, it will be detected easily [28].

One of the well-known blockchain platforms is Ethereum [29]. This platform is especially used in the creation of decentralized open-source applications. Among other advantages, these applications allow us to control the digital value or run a program from anywhere in the world. The Ethereum blockchain network provides great traceability, integrity and transparency of data. Additionally, Ethereum allows us to create smart contracts [30] that specify business logic and agreements between participating nodes in the network. These agreements are executed automatically when certain conditions are satisfied, dealing with the information transmitted across the network or even extracted from other interconnected systems (see Fig. 3).

Smart contracts may need to obtain data placed outside the blockchain. For that reason, blockchain oracles are needed. An oracle is any device or entity that connects the blockchain with off-chain data. These oracles introduce data through external transactions, in this way, we can be sure that the blockchain network contains all the necessary information [28].

Smart contracts can be implemented by using programming languages such as Solidity [31], one of the languages provided by the Ethereum platform. The implementation of a smart contract should be done as efficiently as possible since each of its operations—e.g. a transaction or an execution of the contract—which are performed in the blockchain network may require a certain amount of gas. Gas is the unit that measures the amount of computational effort that is necessary for the execution of a transaction [32].

2.2. Low code

Low-code [33] is a paradigm that allows us to develop solutions without requiring too much knowledge of technologies or programming languages. This is made possible through the use of graphical user interfaces and visual elements. It also provides reusability since there is no need to manually program many of the elements already available by the interface [34]. Low-code is increasingly present in the software development industry when developing products without the need for extensive knowledge of certain technologies [35].

Although low-code is a very new term, its fundamentals are not so new. This paradigm can be considered as a more restrictive derivation of Model-Driven Development (MDD) [36–39], specializing it in certain types of applications such as web or mobile applications [40].

Low-code programming is a process that provides experts in certain technologies a higher level of abstraction by which they automate different steps of the software life cycle. It also allows to speed up the delivery of products and satisfy business needs. It is, therefore, a new approach to software development that enables high scalability and achieves value in a short period of time [41].

Moreover, low-code platforms [42] provide a development environment through a graphical user interface that is used to produce functional applications. This requires some additional programming in certain situations. Low-code platforms greatly reduce the amount of manual programming required, facilitating faster delivery of an application. Additionally, people with limited programming skills have the opportunity to be involved in the development of a large application. Among other advantages, the cost of configuration, training, deployment and maintenance is also reduced [43].

Currently, the development of low-code platforms is increasing around the world. In particular, Node-RED [20] is one of the most popular flow-based tools for visual programming. It is based on the low-code paradigm and allows us to connect the hardware, Application Programming Interfaces (APIs) and services through a graphical user interface tool. Node-RED provides an editor that helps users to create flows by dragging and dropping all available graphical nodes on its tool palette. A flow is a directed graph of processing graphical nodes [44]. While edges establish flow dependencies between the defined graphical nodes, each node performs a part of the computation of a data-driven application [45] when it receives data. Specifically, these graphical nodes are represented by the elements that make up the Node-RED function palette. These elements perform a series of operations and allow the connection of hardware devices, APIs and services, among others. Each node is a piece of code that is implemented in the JavaScript programming language [46]. All available graphical nodes can be easily deployed with a single click. We would like to highlight that the palette provided by Node-RED can be extended with new graphical nodes, as we propose in this paper.

Table 1
Aligning related works and tools with the contributions of this paper.

Approach	C1 (low-code)	C2 (event-driven)	C3 (lightweight devices)	C4 (open source)
Hasan's work [47]	–	–	+/–	+
Singh's work [48]	–	–	–	–
Yong's work [49]	–	–	–	–
Hamdaqa's work [50]	–	+/–	–	+
Li's work [51]	–	–	–	–
Yu's work [52]	–	–	–	–
Li's work [53]	–	–	–	–
Li's work [54]	–	–	+/–	+
Tian's work [55]	–	–	–	–
Settlemint [56]	+/–	–	–	–
Unibright [57]	+	+/–	–	–
Creatorchain [58]	–	+/–	–	–
Aurachain [59]	+	+/–	–	–
Our approach	+	+	+	+

3. Related work

This section discusses the existing proposals and blockchain tools related to our work. In Table 1, we summarize and align existing related works and tools with respect to the contributions of this paper mentioned in Section 1: C1 (integrating blockchain and low-code paradigms), C2 (developing event-driven applications for smart contract management), C3 (deploying the event-driven applications in lightweight devices) and C4 (proposing an open-source solution). Specifically, each of the existing proposals and related tools will be compared according to the contributions made with respect to our approach as follows:

- – : The proposal or tool has not addressed that contribution in its work.
- +/– : The proposal or tool has partially addressed that contribution in its work.
- + : The proposal or tool has addressed that contribution in its work.

Regarding related proposals, Hasan et al. [47] propose a blockchain-based solution for the shipment supply chain management. This proposal makes use of IoT devices, such as sensors integrated with blockchain technology—particularly, the Ethereum public network—to ensure a reliable tracking of such shipments avoiding manipulations. This integration is possible through the use of message brokers with the MQTT protocol [60]. In addition, this proposal uses a lightweight device, namely, Raspberry Pi 3 Model B. Therefore, although Hasan at al.'s work uses sensors and lightweight devices based on blockchain technology, unlike our proposal, it does not provide a low-code platform that allows end users to define in a user-friendly way event-driven applications for the management of smart contracts. Moreover, our low-code approach uses a different and more powerful hardware, namely Raspberry Pi4, which provides lower processing times.

Singh et al. [48] present a system integrating blockchain and Internet of Things (IoT) technologies. The proposal tries to mitigate problems with drug counterfeiting and temperature management in the cold chain. The authors use two different types of blockchain: Hyperledger Sawtooth and Ethereum. Similarly, our proposal is based on the Ethereum blockchain network which brings advantages to our approach such as transparency, immutability, and traceability of data. They also conduct a performance evaluation. However, Singh et al.'s proposal is not tested on lightweight devices. Unlike Singh et al.'s proposal, our proposal provides a user-friendly interface through a low-code platform.

Yong et al. [49] propose an intelligent system based on smart contracts, which are deployed in Ethereum. This system combines blockchain and machine learning for the recommendation, evaluation and forecasting functions on vaccine demand. Nevertheless, unlike our approach, it does not support full automation of the processes of compiling, deploying, interacting and monitoring smart contracts. Yong et al.'s proposal also fail to provide a low-code platform with a user-friendly interface for easily carrying out such processes.

Hamdaqa et al. [50] propose a Domain-Specific Language (DSL) to help software developers create smart contracts and deploy them on a blockchain network. That is, through a graphical tool, software developers will be able to define models that will later generate code for different blockchain platforms. This allows users to abstract which blockchain they are using and what peculiarities each one has. However, it is only able to deploy the contracts on the blockchain network, so it is not able to interact with them or monitor them in real-time. Moreover, unlike our proposal, it is not container-based and its proposal is not intended for low-cost devices.

Li et al. [51] present a solution that seeks a balance between anonymity and traceability in the existing cryptocurrency Monero. In Monero, transactions are made anonymously, however, there is a tracing authority which can revoke that anonymity for misbehavior. In this context, the authors present Traceable Monero that can achieve conditional anonymity and traceability simultaneously. Therefore, Li et al. propose a solution with improved traceability mechanisms with respect to the existing Monero implementation, meeting all security requirements in exchange for a small overhead compared to the existing cryptocurrency. However, they do not propose any low-code approach which allows the development of event-driven applications for smart contract management.

Yu et al. [52] investigate the existing security and privacy issues in IoT by suggesting possible solutions through the use of blockchain technology and Ethereum. Specifically, they demonstrate how blockchain can be integrated with IoT by describing a joint framework. In addition, they show some possible solutions to address the security and privacy issues presented in IoT devices based on blockchain solutions. Moreover, they consider other types of blockchain solutions to cover other needs in addition to security and privacy. However, although Yu et al.'s work uses blockchain technology in conjunction with IoT devices, they do not present a solution that allows the graphical definition of event-driven applications for the smart contract domain.

Li et al. [53] analyze blockchain technology from the point of view of privacy and regulation in cryptocurrencies. Particularly, they investigate existing approaches to enhance privacy in well-known solutions such as Bitcoin and classify different existing methods to put cryptocurrencies under surveillance. Then, they propose two possible solutions to balance privacy and regulation in blockchain-based cryptocurrencies. Therefore, Li et al.'s work does not allow the development of event-driven applications for smart contract management.

In a later work, Li et al. [54] also propose a decentralized voting system based on IoT devices and blockchain technology. In particular, this system satisfies some requirements such as fairness, absence of disputes and voting secrecy. In addition, the protocol implemented by Li et al. has been tested on several devices such as a laptop, a mobile phone and a Raspberry Pi 3 Model B+ through performance and consumption tests. Therefore, although Li et al. use lightweight devices to implement their proposal, unlike our approach, it is not based on the low-code paradigm and does not allow the definition of event-driven applications. In addition, our low-code approach uses a Raspberry Pi4 for the case study, i.e., a more powerful and better-resourced lightweight device than the one used in Li et al.'s work, which will offer processing times lower than those presented by their work.

Tian et al. [55] propose a secure data deduplication and shared auditing scheme based on decentralized storage through blockchain. It aims to achieve space savings while protecting users from losing their data under a single point of failure. In addition, it reduces the cost of metadata calculation and storage through a lightweight authenticator generation algorithm. This scheme achieves a decentralized public audit without the need for third parties that will be shared with all users

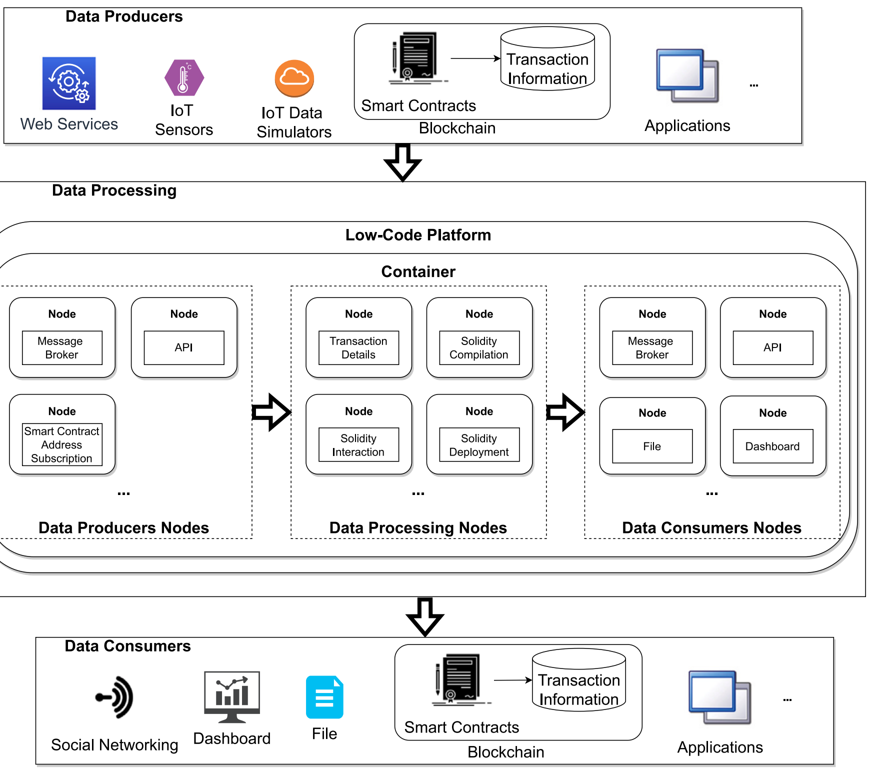


Fig. 4. The proposed approach that integrates blockchain with low-code paradigm.

to avoid this kind of repetitive tasks. Therefore, Tian et al.'s work does not present a low-code approach that allows end users to graphically define event-driven applications for smart contract management.

Therefore, to the best of our knowledge, there are no approaches that integrate blockchain technology with low-code paradigm for facilitating the compilation, deployment, interaction and monitoring of smart contracts, which can be deployed in low-cost devices.

With respect to the existing blockchain tools, Settlemint [56] is a low-code-based Blockchain-Platform-as-a-Service. Its toolkit allows us to build a blockchain network based on other networks such as Ethereum, Polygon or Binance Smart Chain. This tool is focused on enterprise use due to its high costs. Thanks to the use of the low-code paradigm, this tool requires less technical knowledge with respect to other works described in this section, although it is also important to take into account the most frequent errors when using this paradigm [61]. However, our approach has different purposes. Our approach is open-source and does not deploy a blockchain network from scratch; rather, it is a complement to achieve compiling, deploying, interacting and monitoring smart contracts within an existing blockchain network. In addition, our approach allows integration with multiple services and devices.

Unibright [57] is a consulting firm with a team of blockchain specialists with expertise in business process and integration. The provided integration tools are low-code-based. Thanks to this, it allows integrating business processes without the need for code. Its framework is designed to help businesses to integrate blockchain into their processes through its visual tool. Since this tool uses low-code paradigm, it requires less technical knowledge with respect to other works described in this section. However, our approach is more generic, based on an open-source tool that can be extended with new functionalities constantly. Furthermore, our approach also allows the integration of blockchain with a multitude of services and devices without the need for coding.

Creatorchain [58] is a low-code based Blockchain-Platform-as-a-Service. It is based on Polkadot but allows interoperability with other networks such as Ethereum or Binance Smart Chain. It offers a graphical interface for developers and also for users depending on the purpose for which the tool is used. Creatorchain enables the graphical definition, compilation and deployment of smart contracts. However, it does not allow integration of the blockchain network with other existing services or devices. In addition, although its cost is lower than the other tools presented, its use requires costs.

Aurachain [59] is a low-code platform that enables fast software development. It is a tool focused on developers as well as citizens through the use of a visual module with drag and drop functions. In addition, it allows the creation of deployable blockchain applications on Hyperledger, Ethereum and other compatible networks. Aurachain also allows the integration of its applications with other systems and services. Similarly, this tool requires less technical knowledge compared to other mentioned works since it is based on the low-code paradigm. However, the use of this tool entails high costs that users and many companies cannot afford to implement blockchain-based solutions.

4. The EDALoCo approach

This section presents our low-code approach for developing event-driven applications for user-friendly smart contract management. An overview of our proposed architecture, which is composed of three main layers (Data Producers, Data Processing and Data Consumers), is illustrated in Fig. 4.

4.1. *Data producers*

The **Data Producers layer** is composed of Web Services, IoT sensors, IoT data simulators, smart contracts and applications, among others, which generate data relevant to our approach.

Web Services are a means of intercommunication and interoperability between machines connected in a network. IoT sensors can be manufactured by several suppliers and can measure different parameters, such as temperature, humidity or air quality. Applications are a type of computer software designed to perform a group of coordinated functions, tasks or activities for the benefit of the user.

The blockchain side is composed of smart contracts that have already been deployed in a blockchain network. When a smart contract is deployed or one of its functions is invoked, a transaction is created. Transactions are grouped together to create blocks. All transactions registered in the blockchain network can be monitored in real time through our proposal.

As shown in Fig. 4, the blockchain network contains information about the transactions done such as destination address, transaction hash, or gas cost. These data can be automatically consumed by some nodes in our proposal, as explained in Section 4.2.

These data producers act as an oracle, i.e., it is an intermediary between the real world and the smart contract. Specifically, data from different data producers are received and could be preprocessed to convert them into a suitable format. Depending on the device or service supplier, the received data may have different structures. Thereby, these data could be transformed and unified to a single format that will be necessary to be sent to the low-code platform (Data Processing Layer), i.e. data in different formats such as Comma-Separated Values (CSV) and Extensible Markup Language (XML) are transformed into a unique format: JavaScript Object Notation (JSON).

4.2. Data processing

The **Data Processing layer** is composed of a software container that hosts a low-code platform.

The low-code platform facilitates the developing of event-driven applications. To this end, this platform provides a browser-based editor that allows for the user-friendly developing of these applications through node flows. Nodes can be grouped according to their functionalities: (i) Data Producer nodes, which are responsible for receiving data from the Data Producers layer and sending it to the Data Processing nodes; (ii) Data Processing nodes, which are in charge of data processing such as, compiling, deploying or interacting with a smart contract, as well as sending the produced results to the Data Consumer nodes; and (iii) Data Consumer nodes, which allow for connecting and sending data from our low-code platform to the Data Consumers layer, which is composed of several elements such as dashboards, files and the blockchain network.

We decided to base our approach on the Node-RED low-code platform because it brings us many advantages: (i) it can be run in software containers, resulting in a portable solution that can be used in any machine regardless of aspects such as configuration or operating system; (ii) it can be executed on most lightweight devices so any person or entity without the need for many resources can use our proposal; (iii) it allows to develop event-driven applications collaboratively; and (iv) it allows easy import and export of created flows.

More specifically, we have implemented an extension of the Node-RED flow-based tool that facilitates the integration of blockchain technologies with the low-code paradigm. The nodes developed for the extension of the Node-RED low-code platform's palette have specific functionalities, as detailed in the following subsections. These nodes, which we implemented using NodeJS, JavaScript and HTML as suggested by the Node-RED documentation, were included in the Node-RED editor palette to make them available to end users (see Fig. 5). Although in this work we use it for a specific case study, it is important to remark that these nodes are generic and can be used for all smart contracts defined in several blockchain networks. That way, users will be able to graphically develop event-driven applications to manage smart contracts in a blockchain network. Thanks to the transparency provided by the use of some blockchain networks, such as, Ethereum, we do not need any kind of permission for anyone to be able to monitor all the transactions of a deployed smart contract.

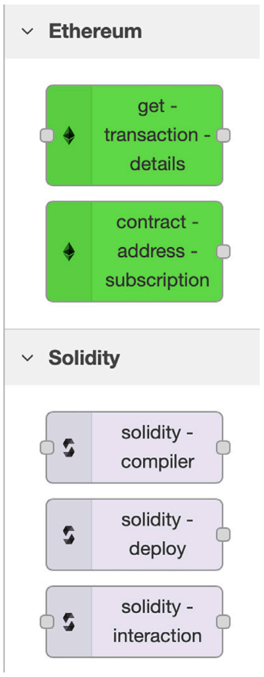


Fig. 5. Screenshot of the Node-RED tool palette extended with blockchain support.

4.2.1. Smart contract address subscription

We have developed a new node, called *Smart Contract Address Subscription*, in Node-RED. It is a listener that receives all hashes of new transactions mined on the selected blockchain network from a specific public address of a smart contract. This node has no inputs.

Its parameters are as follows: (i) *Web Socket*, which indicates the blockchain network where the specified smart contract is already deployed, and (ii) *Contact Address*, which indicates the address of the smart contract that is deployed on the blockchain network and to which we wish to subscribe.

The output of this node is the result of the smart contract subscription process. If the subscription has been successful, then all transactions to or from the specified contract address will be displayed.

4.2.2. Transaction details

We have developed a new node, called *Transaction Details*, in Node-RED. It gets all the details of a transaction that is already mined on the blockchain network from its hash. The input of this node is the hash of the transaction to be queried in the blockchain network.

Its parameters are as follows: (i) *Web Socket*, which indicates the blockchain network on which the transaction has taken place, and (ii) *Detail*, which indicates what details to be obtained from the queried transaction. It can be a single piece of information such as gas used, source address or destination address. However, it is also possible to query all the details of the transaction.

The output of this node is the result of the transaction query process. If the query has been successful, then the details selected in the node will be displayed.

4.2.3. Solidity compiler

We have developed a new node, called *Solidity Compiler*, in Node-RED. It allows to check that the developed Solidity code is correct and has no errors. The input of this node is the Solidity code to be compiled.

Its parameter is as follows: *Smart Contract Name*, which indicates the name of the smart contract to be compiled.

The output of this node is the result of the Solidity code compilation process. In case of an error in the compilation, the error reasons will be shown. If the compilation has been successful, then detailed information about all the artifacts resulting from the compilation will be displayed.

4.2.4. Solidity deploy

We have developed a new node, called *Solidity Deploy*, in Node-RED. It deploys the smart contract on the blockchain network chosen by the user. This contract must not have any compilation errors. This node has no inputs.

Its parameters are as follows: (i) *Web Socket*, which indicates the blockchain network where the Solidity code will be deployed. (ii) *Bytecode*, which indicates the compiled code necessary for the correct deployment of the Solidity code. It is the hexadecimal representation of the compiled contract to object code. This bytecode can be obtained through compilation using the Solidity Compiler node. (iii) *ABI* (Application Binary Interface), which indicates a smart contract interface required to be able to deploy it correctly. This allows interacting with the smart contract from outside the network or contract–contract interactions. This ABI can be obtained through compilation using the Solidity Compiler node. (iv) *Public Address*, which indicates the public address of the account that will be in charge of deploying the smart contract on the blockchain network. This account can be considered the "owner" of the smart contract, and (v) *Private Key*, which indicates the private key of the account that will be in charge of deploying the smart contract on the blockchain network. This account can be considered the "owner" of the smart contract.

The output of this node is the result of the Solidity code deployment process. If the deployment has been successful, then the address of the smart contract assigned in the blockchain network will be displayed.

4.2.5. Solidity interaction

We have developed a new node, called *Solidity Interaction*, in Node-RED. It invokes the functions of the smart contract already deployed on the blockchain network.

The input of this node is the data with the necessary parameters to interact with the desired smart contract function. Note that not all smart contract functions need parameters.

Its parameters are as follows: (i) *Web Socket*, which indicates the blockchain network where the specified smart contract is already deployed. (ii) *Private Key*, which indicates the private key of the account that will be in charge of interacting with the smart contract already deployed on the blockchain network. (iii) *Smart Contact ABI*, which indicates a smart contract interface required to be able to interact correctly. This allows interacting with the smart contract from outside the network or contract-contract interactions. This ABI can be obtained through compilation using the Solidity Compiler node, and (iv) *Smart Contact Address*, which indicates the address of the smart contract that is deployed on the blockchain network and with which we want to interact. This address can be obtained through deployment using the Solidity Deploy node.

The output of this node is the result of the smart contract interaction process. If the interaction has been successful, then all details of the completed transaction will be displayed.

4.3. Data consumers

The **Data Consumers layer** is composed of social networking, dashboards, files, smart contracts, databases and applications, which consume data from our approach. Optionally, multiple devices and services can consume data from the same event-driven application.

The blockchain side is composed of smart contracts that can be deployed on the blockchain network or have been already deployed. When a smart contract is deployed or one of its functions is invoked through our approach, a transaction is created and its status in the blockchain network changes.

Social Networking refers to the use of social networks as a means of sharing data and results. A dashboard is a way to represent the data and results obtained in a user-friendly way. Finally, File refers to the storage in some local or remote directory of the data and results obtained.

Fig. 6. COMET IoT wireless temperature device.

These data consumers will be in charge of transferring all the results of our approach to other systems or tools. In other words, they act as intermediaries between our approach and the final systems. This generic approach based on a low-code platform allows to be integrated into business processes and to be part of a larger system thanks to its modular and lightweight architecture.

5. Case study

To demonstrate the feasibility of our low-code proposal, we applied it to a real-world IoT case study, as presented below.

5.1. Description

This case study was implemented by using Ethereum as the blockchain network, a smart sensor as the IoT device and a lightweight device as a host for the low-code platform container, as detailed below. Note that other sensors, devices, services and smart contracts could be added to the architecture, if necessary.

To easily monitor temperatures of different vaccines in real time, the Data Producers layer is composed of an IoT wireless temperature data logger with built-in sensor and Global System for Mobile communications (GSM) modem. This device (see Fig. 6), manufactured by Comet System [62], fulfills the requirements of EN ISO/IEC 17025 standard, which is very useful for performing testing, sampling or calibration and obtaining reliable results [63]. This device can send Short Message Service (SMS) and JSON messages using the General Packet Radio Service (GPRS) [64] technology. The measured temperature data can be automatically sent to a data server in real time and also recorded in non-volatile electronic memory to be transferred to a PC. Some of the most important features of this device are as follows:

- Temperature measuring range: −20 to +60 °C.
- Accuracy of temperature measurement: ±0.4 °C.
- Total memory capacity: 500 000 values.
- Dimensions: 61 × 93 × 53, with antenna 120 × 93 × 53.
- Weight: 260 g.
- Battery: SONY LiIon 5200 mAh.

Particularly, this sensor measures the temperature every 5 min (this frequency can be configured) and then sends it to a message broker. These measurements are received by our event-driven application through data source nodes that connect to the message broker and consume in real time all the temperatures emitted by the sensor.

The Blockchain component, proposed in the Data Producers layer (see Fig. 4), is composed of two smart contracts, called

AstraZenecaVaccine and ModernVaccine, which we implemented in Solidity for defining the logic necessary to register and detect when the temperature exposed by an AstraZeneca vaccine [65] or Moderna vaccine [66] is out of the allowed range. The code of these smart contracts can be downloaded from [67]. Both contracts have the same functions with minor differences in terms of temperature thresholds. Specifically, the contract functions are as follows:

- registerTemperature: this function is in charge of receiving the data to be registered in the blockchain network. In addition, the received temperature is checked against set thresholds. If the temperature is not within the thresholds, then a temperature warning is generated. In the case of the AstraZeneca vaccine, if the temperature is lower than 2 °C or higher than 8 °C, the temperature will be recorded as a temperature outside the set thresholds. However, if the vaccine is Moderna, the established thresholds will be between −25 °C and −15 °C. These temperature thresholds are the ones recommended for correct conservation of the vaccines according to the European Medicines Agency suppliers [65,66]. Since this function writes data into the blockchain network, it previously requires the approval of the majority of the nodes in the network. That is, registering this data in the blockchain network will have an associated cost which will depend on the number of operations that are performed within the function.
- getTemperatureReading: this function allows to get all the data that has been registered so far in the blockchain network by the register Temperature function. As this operation is only a query and does not require approval by the blockchain network, it does not have any gas cost.
- getTemperatureWarning: all the temperature warnings that have been registered in the blockchain network by the registerTemperature function are obtained. These temperature warnings will give us all the details of the recorded temperatures, which sensor recorded them and at what time they occurred. As this operation is only a query and does not require approval by the blockchain network, it does not have any gas cost.

Thereby, these smart contracts can record in the blockchain network the following information emitted by the sensor:

- **timestamp**: the timestamp value at which the temperature was measured by the sensor.
- **sensorId**: an id that uniquely identifies a device that is in charge of taking and sending temperatures.
- **temperature**: a temperature value taken at a specific time and sent by the sensor.

All transactions recorded in the blockchain network will take place transparently and immutably. Moreover, the results can be queried by anyone, anywhere in the world at any time.

The Data Processing layer is made up of a lightweight device to deploy our architecture, in particular, it is a Raspberry Pi4. Currently, this is the latest model available that offers an improvement in speed and performance over previous models. This model is also quiet and has low power consumption compared to other devices [23]. Note that the use of this type of device in the case study is to demonstrate that the proposal can be deployed on lightweight devices. However, it can be also deployed on computers and more powerful devices. Some of the most important features of the Raspberry Pi4 are:

- Broadcom BCM2711, Quad core Cortex-A72 (ARM v8) 64-bit SoC @ 1.5 GHz.
- 8 GB LPDDR4-3200 SDRAM.
- Gigabit Ethernet.
- Micro-SD card slot for loading operating system and data storage.
- Operating temperature: 0–50 °C.

This device will be in charge of hosting the container necessary for the correct operation of the approach. The container used in this proposal has been managed using the Docker tool [68].

The container is responsible for hosting the low-code platform. Through this platform, end-users can drag and drop different nodes available in the editor palette. As mentioned in Section 4.2, these nodes can have different functions, such as data sources, data processors, or data consumers.

This low-code platform provides an editor where flows for creating an event-driven application can be graphically developed. The data source nodes of the application to be modeled can be of various types, for example, a source node that supports an MQTT protocol client that receives the formatted data from the smart sensor.

Optionally, processing nodes can be modeled. These nodes have a data input and a data output. These nodes execute a series of functions with the data received. The data obtained as a result of executing these functions will be sent to the other nodes. Specifically, in this proposal, the received data are deployed in the Ethereum blockchain network. This action generates some results that are the details of the deployed transaction to be sent to the output nodes.

The output nodes allow us to store data, send data or display data. Remarkably, a processing node can be linked to several output nodes, so the same information will be sent, stored or displayed in different places. In our proposal, the tests performed have been sent to a message broker by using the Kafka protocol, storing the data in a file and displaying the results on a dashboard.

5.2. Modeling event-driven applications

Four event-driven applications for managing smart contracts have been graphically modeled by using our blockchain-based graphical tool (see Section 4), as explained below.

Fig. 7 depicts one of such modeled event-driven applications. Particularly, this application, which has been modeled with 10 nodes, allows end-users to compile a given smart contract. The first node (*Solidity Code*) corresponds to a data producer, which will contain the Solidity code to be compiled. This node is connected to the second node (*Solidity Compiler*), a data processor responsible for transparently compiling the Solidity code injected from the data producer. Note that this node requires a certain configuration in the form of parameters to be specified by the user, as illustrated in Fig. 8. The only mandatory parameter required by the node is the name of the smart contract to be compiled. Then, the data processor node is linked to several data consumer nodes where the data processing results will be published. Different types of data consumer nodes can be used such as message broker, file and on a dashboard. Note that all data typed by the user in each node is validated, so, an error will be displayed for correction if the entered data is incorrect.

Specifically, the *kafka-producer* node is in charge of publishing the compilation results in a message queue. The *write file* node saves the compilation results in a file. The rest of the orange and blue nodes are in charge of obtaining the relevant data from the compilation results and displaying them through a dashboard.

Fig. 9 shows the modeled event-driven application for deploying a smart contract. The first node (*Solidity Deploy*) is a data producer that allows end-users to deploy a smart contract on a blockchain network in a user-friendly way. This node requires a previous configuration before deploying a smart contract correctly. Fig. 10 shows the required data to be provided by the user, such as web socket, bytecode, ABI, public address and private key. Web socket is the specific URL address needed to properly connect to the blockchain network (in our case, the Ethereum network). Bytecode and ABI specify the information about the details of the contract to be deployed. The public address and private key are from the user who will deploy the contract on the Ethereum blockchain network. Note that many of these fields can be obtained through the use of other nodes, such as the compilation node.

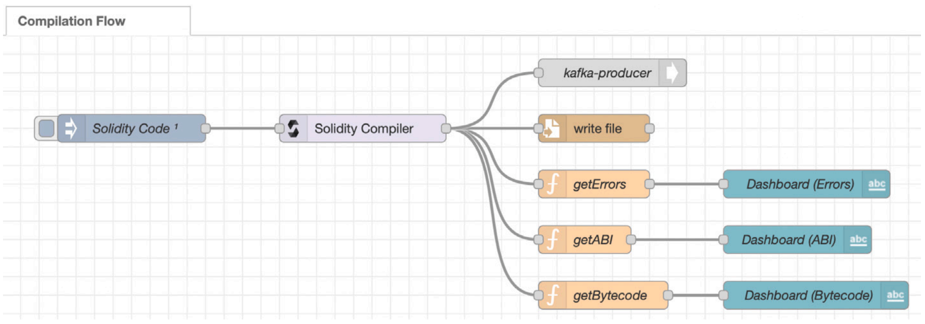
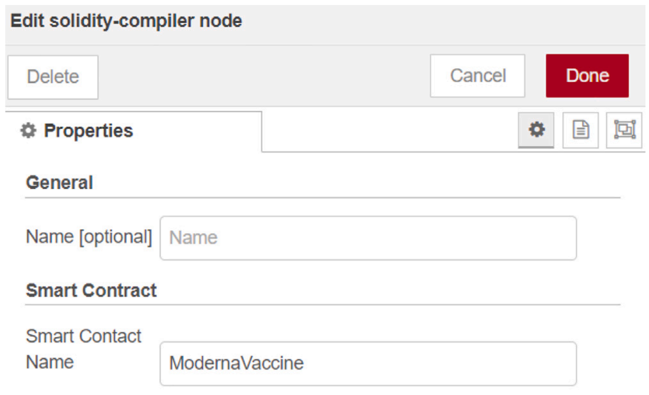


Fig. 7. Node-RED flow for compiling a smart contract.


Edit solidity-compiler node

Delete | Cancel | Done

Properties

General

Name [optional]: Name

Smart Contract

Smart Contact Name: ModernaVaccine

Fig. 8. Configuration details of the *Solidity Compiler* node.

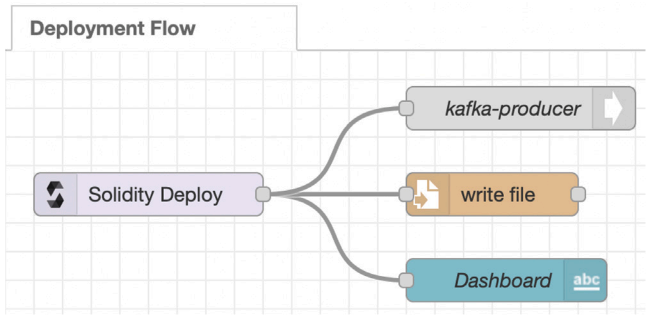
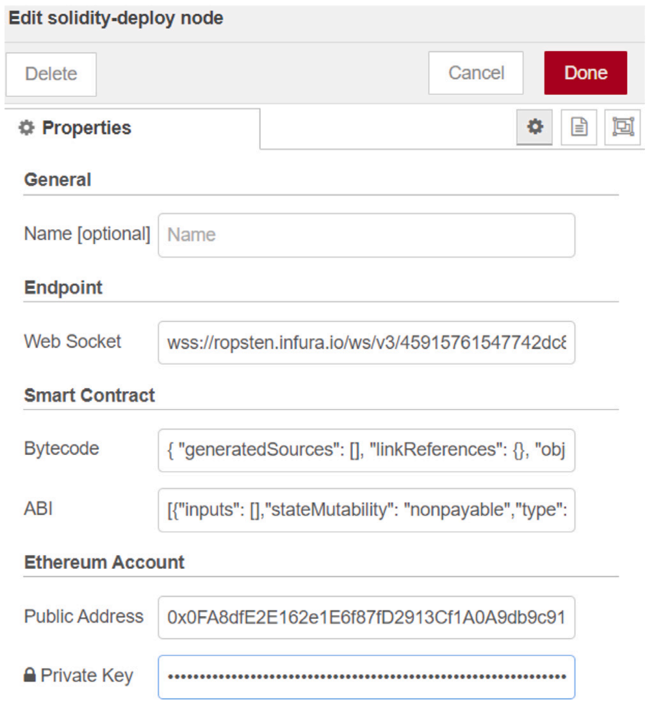


Fig. 9. Node-RED flow for deploying a smart contract.


Edit solidity-deploy node

Delete | Cancel | Done

Properties

General

Name [optional]: Name

Endpoint

Web Socket: wss://ropsten.infura.io/ws/v3/45915761547742dc

Smart Contract

Bytecode: { "generatedSources": [], "linkReferences": {}, "obj

ABI: [{"inputs": [],"stateMutability": "nonpayable","type":

Ethereum Account

Public Address: 0x0FA8dfE2E162e1E6f87fD2913Cf1A0A9db9c91

Private Key: ••

Fig. 10. Configuration details of the *Solidity Deploy* node.

The rest of the nodes represent data outputs that will be responsible for recording the results in different places such as message brokers, files and on a dashboard. In this particular application flow, and unlike the previous one, there are no data processing nodes. Therefore, data are produced and stored in the blockchain without being modified by any node.

In particular, the *kafka-producer* node is in charge of publishing the deployment results in a message queue. The *write file* node saves the deployment results in a file. Then, the *Dashboard* node is in charge of displaying the deployment result through a dashboard.

The third application flow for facilitating the interaction with a blockchain network is formed by 26 nodes, as shown in Fig. 11. The first node (*MQTT*) is a message broker–client that receives the data to be emitted by the sensor and preprocessed before reaching the low-code platform. These transmitted data use the MQTT protocol. Once the data have been received by a processing node, these will be automatically registered in the blockchain network. To this end, the node must be previously configured, as depicted in Fig. 12. Among others, the following data are required to the user: web socket, private key, smart contract ABI and smart contract address. The private key is from the user who will interact with the contract deployed in the blockchain network. ABI and address specify the information about the details of the contract already deployed. After interacting with the blockchain network, the processed results are received by data consumer nodes, such as message broker, file and on a dashboard.

More specifically, the *kafka-producer* node is in charge of publishing the interaction results in a message queue. The *write file* node saves the interaction results in a file. The rest of the orange and blue nodes are in charge of obtaining the relevant data from the interaction results and displaying them through a dashboard.

The event-driven application modeled for monitoring a smart contract deployed in a blockchain network is composed of 22 nodes (see Fig. 13). The first node (*Contract_Address Subscription*) is a data producer which subscribes to a smart contract already deployed in a blockchain network. This node is listening in real time to the blockchain network to notify the new mined transactions whose source or destination is such a smart contract. This node requires a previous configuration, as illustrated in Fig. 14, in particular, web socket and contract address. The contract address is the public address of the smart contract that is deployed on the blockchain network and which is to be monitored in real time. Then, the *Get Transaction Details* node receives

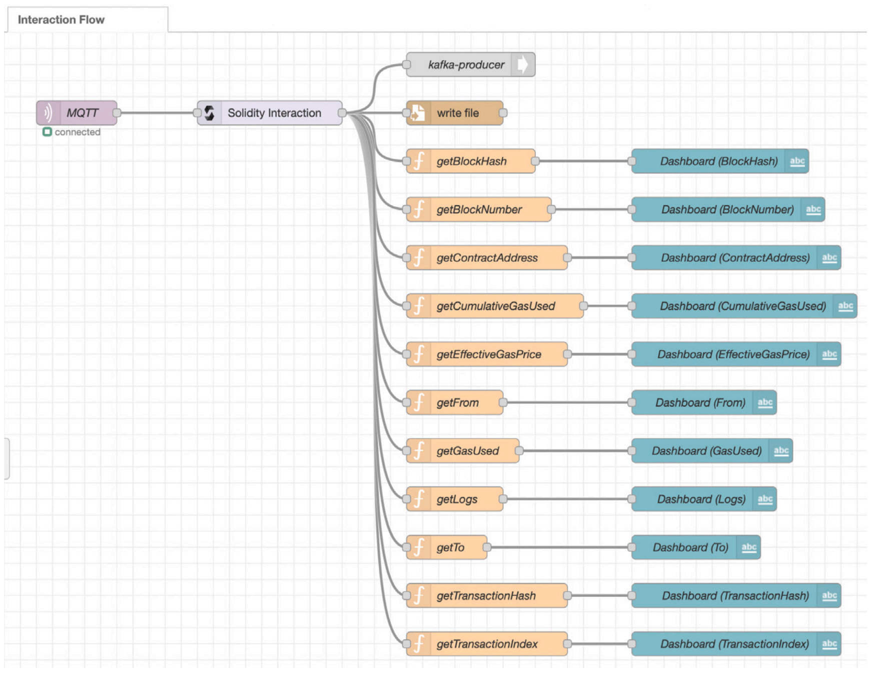
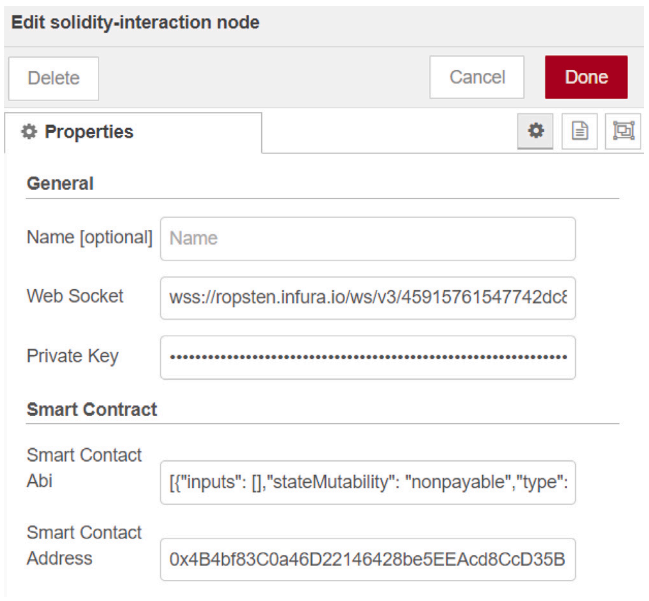


Fig. 11. Node-RED flow for interacting with a smart contract.



General

Name [optional]	Name
Web Socket	wss://ropsten.infura.io/ws/v3/45915761547742dc8
Private Key	••

Smart Contract

Smart Contact Abi	[{"inputs": [],"stateMutability": "nonpayable","type":
Smart Contact Address	0x4B4bf83C0a46D22146428be5EEAcd8CcD35B

Fig. 12. Configuration details of the *Solidity Interaction* node.

the result of the (*Contract_Address Subscription*) node. From the hash of the monitored transactions in real time, this node can query the details of the monitored transactions. This node also requires a previous configuration so that it can properly connect to the blockchain network and then query the needed details. Fig. 15 shows the data required for its correct configuration, i.e, Web socket and the details to be obtained. Concretely, details refer to the desired information to be obtained from the transactions listened to in real time; for instance, all available information or just the transaction number or gas cost. The data consumer nodes (message broker, file and display) will receive the details of all monitored transactions in real time.

Note that the *kafka-producer* node is in charge of publishing the monitoring results in a message queue. The *write file* node saves the monitoring results in a file. The rest of the orange and blue nodes are in charge of obtaining the relevant data from the monitoring results and displaying them through a dashboard.

5.3. Executing the modeled event-driven applications

After modeling the event-driven applications for compiling, deploying, interacting and monitoring smart contracts, these were executed by obtaining the following information.

Fig. 16 shows the results obtained upon the *Compilation* flow execution. Specifically, it shows the information necessary for the deployment flow, such as the ABI and bytecode of the smart contract. Among others, it also shows if there are any errors.

The output obtained for the execution of the *Deployment* flow is illustrated in Fig. 17. Particularly, it presents two messages with relevant information about the deployment process. Firstly, it shows that it is trying to deploy the contract and which user is deploying it. Then, it indicates that the contract has been successfully deployed and the address that has been assigned to this contract.

After executing the *Interaction* flow, the obtained output can be checked in Fig. 18. In particular, all information about the transaction that has just been mined is displayed. As an example, the block in which the transaction has been included, the hash of the transaction or the gas used.

Fig. 19 depicts the results obtained as a result of executing the *Monitoring* flow. It outputs the information of the transaction listened in real time whose source or destination is the public address that has been previously configured; among other information, the transaction hash, block hash or block number.

Note that all above results have been displayed on a dashboard, replicated and saved in a file and sent to a message broker.

6. Evaluation

This section presents the evaluation of our EDALoCo proposal, which we have applied to the case study described in Section 5.

EDALoCo can be considered accessible for software developers who are non-experts in blockchain since it is based on the low-code paradigm, which provides a great level of user-focus by abstracting from technical details [69].

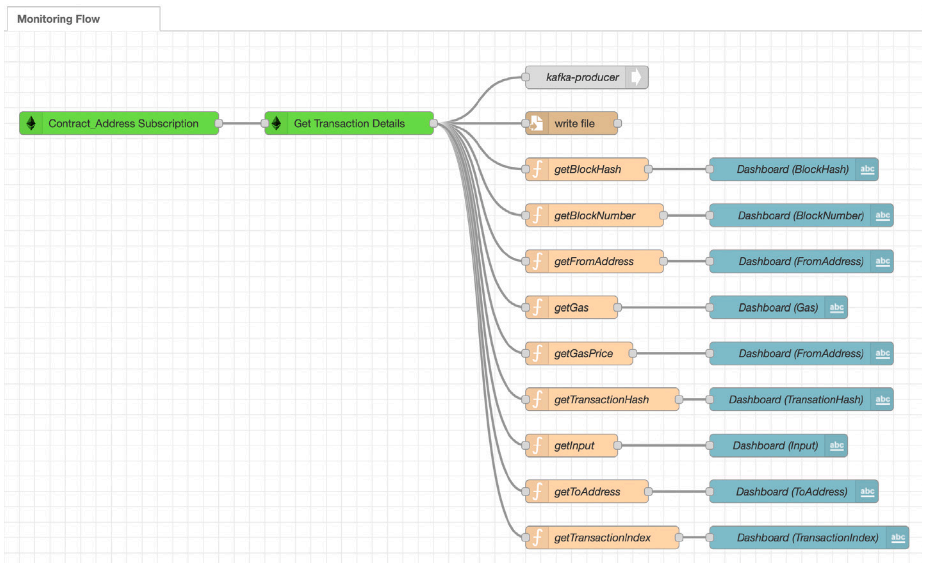
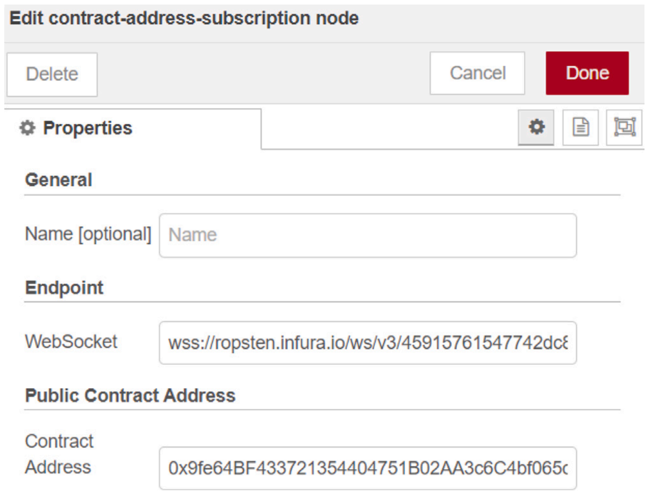


Fig. 13. Node-RED flow for monitoring a smart contract.


Edit contract-address-subscription node

Delete | Cancel | Done

Properties

General

Name [optional]: Name

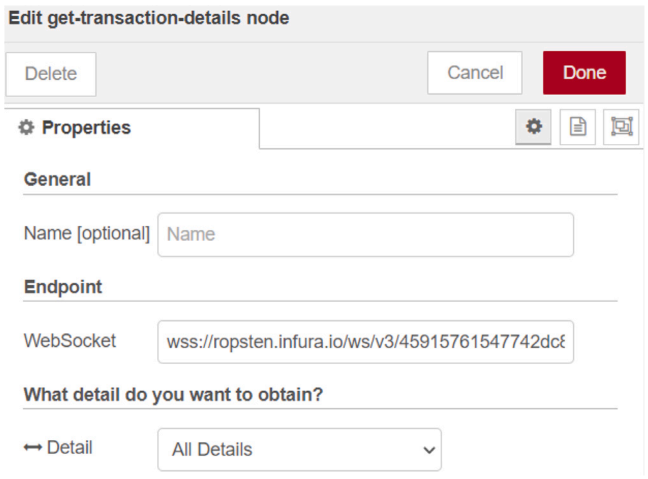
Endpoint

WebSocket: wss://ropsten.infura.io/ws/v3/45915761547742dc8

Public Contract Address

Contract Address: 0x9fe64BF433721354404751B02AA3c6C4bf065c

Fig. 14. Configuration details of the *Contract Subscription* node.


Edit get-transaction-details node

Delete | Cancel | Done

Properties

General

Name [optional]: Name

Endpoint

WebSocket: wss://ropsten.infura.io/ws/v3/45915761547742dc8

What detail do you want to obtain?

Detail: All Details

Fig. 15. Configuration details of the *Transaction Details* node.

Table 2
Public addresses of the smart contracts deployed in the Ethereum blockchain network.

Contract name	Contract address
ModernaVaccine	0×9fe64BF433721354404751B02AA3c6C4bf065cEc
AstraZenecaVaccine	0×4B4bf83C0a46D22146428be5EEAcd8CcD35B1b17

6.1. Functionality evaluation

To evaluate the functionality of our proposal, we ran the case study during 1 week (from 02/06/2021 to 09/06/2021), considering the two mentioned application scenarios: Moderna and AstraZeneca vaccines (see Section 5).

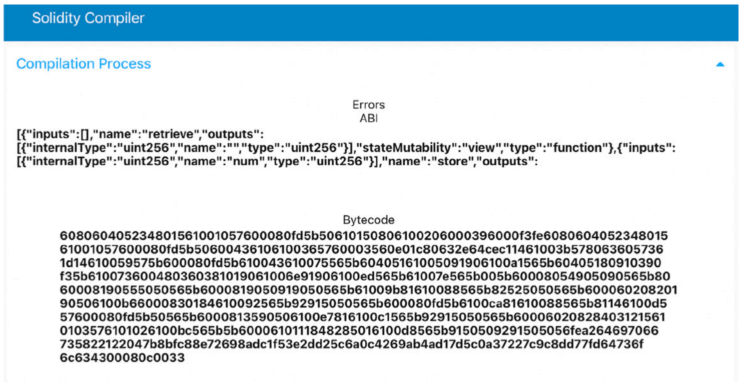
For the Moderna scenario, we placed the temperature sensor in a freezer. This vaccine was kept at a temperature with thresholds between −25 °C and −15 °C. Fig. 20 shows all temperatures measured by the sensor every 5 min.

For the AstraZeneca scenario, we placed the temperature sensor in a refrigerator. This vaccine was kept at a temperature with thresholds between 2 °C and 8 °C. All temperatures measured by the sensor every 300 s can be seen in Fig. 21.

To automatically store such temperature measurements in the blockchain to make them visible to anyone around the world, we deployed the two described smart contracts in the Ethereum network. As a consequence, a unique public address was automatically assigned to each contract (see Table 2).

Therefore, thanks to the transparency, traceability and immutability of the Ethereum blockchain network, all transactions mined on the network, i.e. all recorded temperatures of each smart contract can be queried by anyone transparently. Using the Node-RED tool, which we have extended with the blockchain support, all these temperatures have been registered by non-experts in blockchain (see Section 5). Fig. 22 shows an excerpt of a query on the conducted transactions of the Moderna vaccine smart contract, while Fig. 23 shows another example of a query on AstraZeneca vaccine smart contract's transactions.

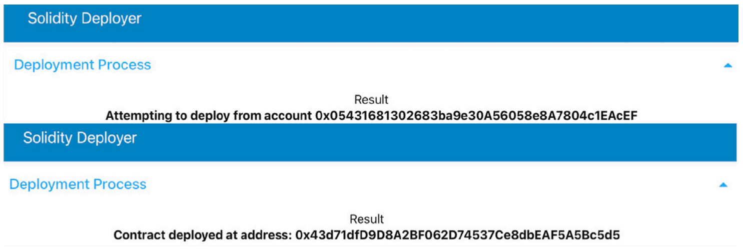
As a result, Table 3 summarizes the information on the vaccine smart contracts. Particularly, it presents the temperature measurements registered in the Ethereum blockchain network. Additionally, it shows the detected temperature warnings also registered in the Ethereum network, i.e., the temperatures outside the established thresholds. Moreover, the total cost in gas for having mined all these transactions and the average cost of gas for each transaction are provided in Table 3.



```
Errors
ABI
[{"inputs":[],"name":"retrieve","outputs":
[{"internalType":"uint256","name":"","type":"uint256"}],"stateMutability":"view","type":"function"},{"inputs":
[{"internalType":"uint256","name":"num","type":"uint256"}],"name":"store","outputs":

Bytecode
6080604052348015610010576000080fd5b50610150806100206000396000f3fe6080604052348015
61001057600080fd5b506004361061003657600003560e01c80632e64cec11461003b578063605736
1d14610059575b600080fd5b610043610075565b604051610050919061000a1565b60405180910390
f35b61007360048036038101906100 6e91906100ed565b61007e565b005b60008054905090565b80
600081905550505650565b6000819050919050565b61009b81610088565b82525050565b600060208201
90506100b6600083018461009 2565b92915050565b600080fd5b6100ca81610088565b81146100d5
57600080fd5b50565b6000813590506100e7816100c1565b92915050565b6000602082840312 1561
0103576101026100bc565b5b60006101111848285016100d8565b9150509291505056fea264697066
735822122047b8bfc88e72698adc1f53e2dd25c6a0c4269ab4ad17d5c0a37227c9c8dd77fd64736f
6c634300080c0033
```

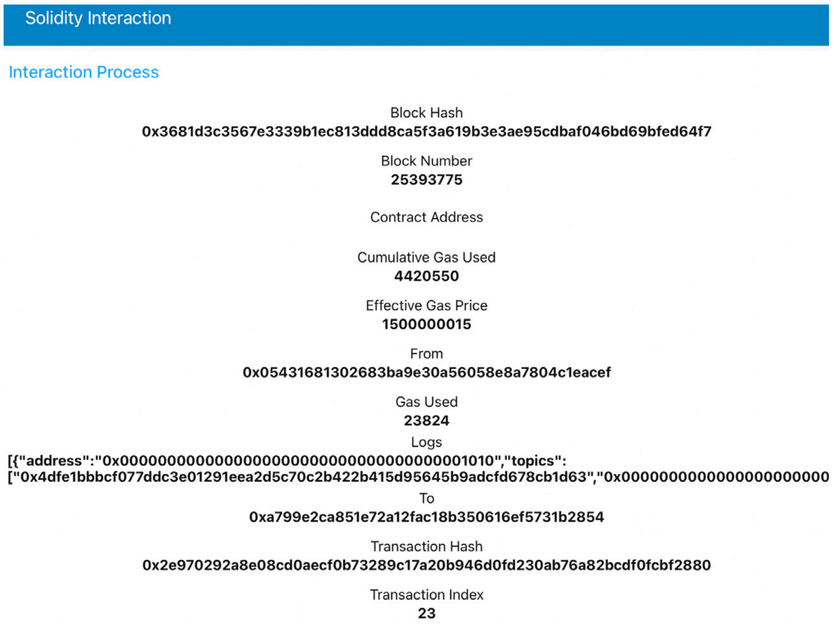
Fig. 16. Dashboard with the results obtained after executing the *Compilation* flow.



```
Result
Attempting to deploy from account 0x05431681302683ba9e30A56058e8A7804c1EAcEF

Result
Contract deployed at address: 0x43d71dfD9D8A2BF062D74537Ce8dbEAF5A5Bc5d5
```

Fig. 17. Dashboard with the results obtained after executing the *Deployment* flow.



```
Block Hash
0x3681d3c3567e3339b1ec813ddd8ca5f3a619b3e3ae95cdbaf046bd69bfed64f7
Block Number
25393775
Contract Address

Cumulative Gas Used
4420550
Effective Gas Price
1500000015
From
0x05431681302683ba9e30a56058e8a7804c1eacef
Gas Used
23824
Logs
[{"address":"0x0000000000000000000000000000000000001010","topics":
["0x4dfe1bbbcf077ddc3e01291eea2d5c70c2b422b415d95645b9adcfd678cb1d63","0x0000000000000000000000
To
0xa799e2ca851e72a12fac18b350616ef5731b2854
Transaction Hash
0x2e970292a8e08cd0aecf0b73289c17a20b946d0fd230ab76a82bcdf0fcbf2880
Transaction Index
23
```

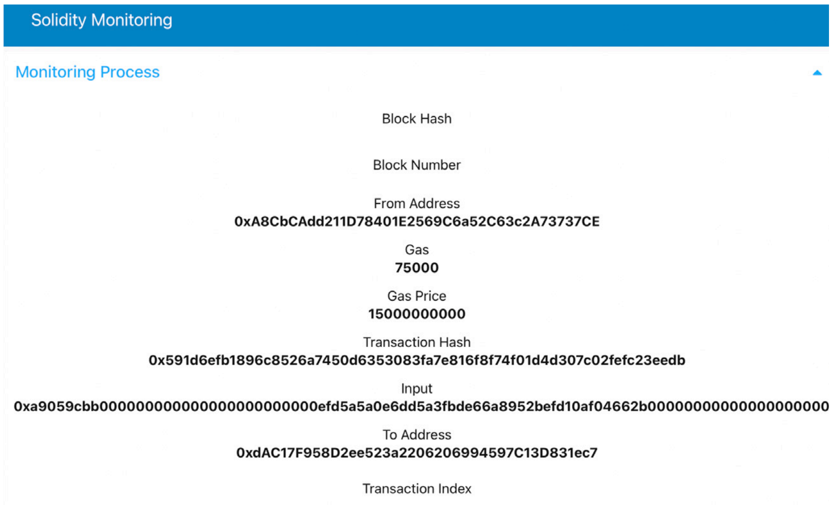
Fig. 18. Dashboard with the results obtained after executing the *Interaction* flow.

Table 3
Summary of the relevant data for the deployed smart contracts.

Smart contract	Mined transactions	Detected warnings	Total gas used	Average gas used
ModernaVaccine	716	3	69252186	96720.93016759776
AstraZenecaVaccine	501	1	48343418	96493.84830339321

Therefore, we can affirm that the flows modeled in Section 5 were correctly deployed. Moreover, thanks to the use of the Ethereum network, the registered data are transparent, immutable and traceable.

Finally, after having performed the evaluation, we can state that software developers can graphically develop event-driven applications that allows managing smart contracts in a blockchain network. All


Solidity Monitoring

Monitoring Process

Block Hash

Block Number

From Address
0xA8CbCAdd211D78401E2569C6a52C63c2A73737CE

Gas
75000

Gas Price
15000000000

Transaction Hash
0x591d6efb1896c8526a7450d6353083fa7e816f8f74f01d4d307c02fefc23eedb

Input
0xa9059cbb000000000000000000000000efd5a5a0e6dd5a3fbde66a8952befd10af04662b00000000000000000000

To Address
0xdAC17F958D2ee523a2206206994597C13D831ec7

Transaction Index

Fig. 19. Dashboard with the results obtained after executing the *Monitoring* flow.

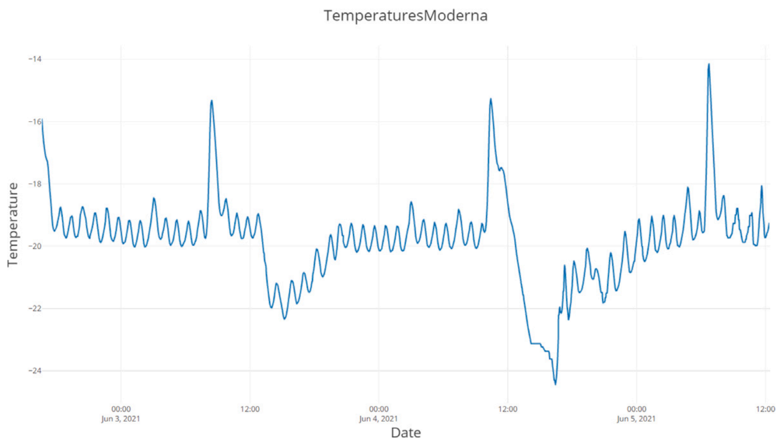


Fig. 20. Recorded temperatures of the Moderna vaccine.

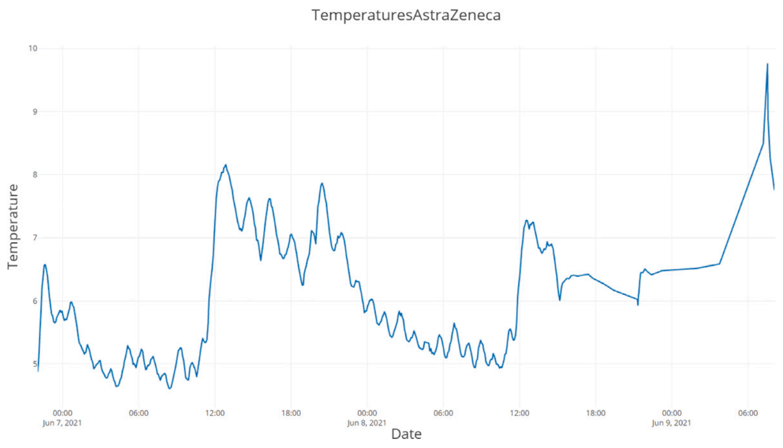
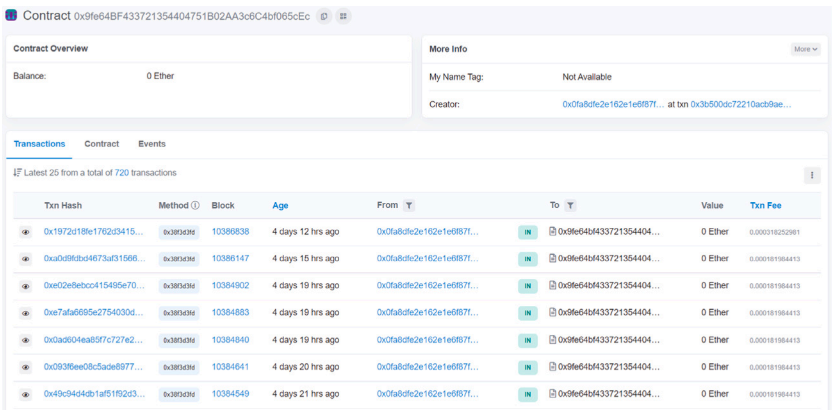


Fig. 21. Recorded temperatures of the AstraZeneca vaccine.

this is achieved using the low-code paradigm, making the approach accessible to non-experts by avoiding repetitive programming tasks. Moreover, thanks to our extension of the Node-RED tool that allows defining event-driven applications through graphical flows, all temperatures measured in real time by devices and services will be registered in the blockchain network automatically and without the need for human interaction.

6.2. *Performance evaluation*

We conducted a performance evaluation of our proposal deployed in a Raspberry Pi. We used the RPi-Monitor [70] software to monitor some system metrics of such a low-cost device. In addition, it provides a web server that allows us to display the current status of the device as



Contract 0x9fe64BF433721354404751B02AA3c6C4bf065cEc

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

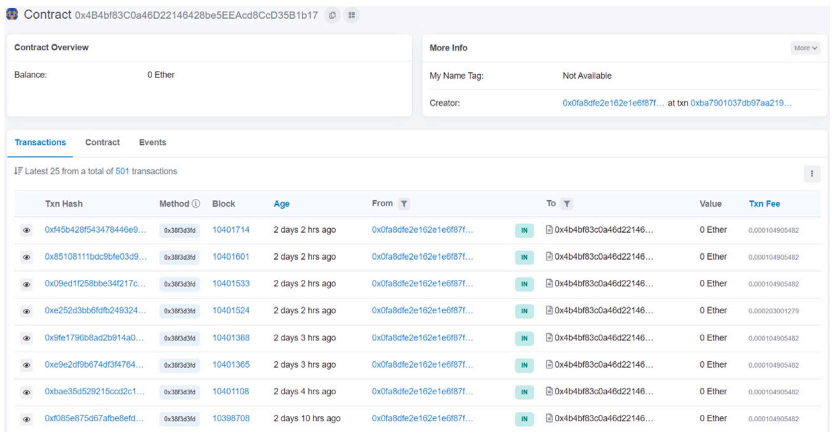
Creator: 0x0fa8dfe2e162e1e6f87f… at txn 0x3b500dc72210acb9ae…

Transactions | Contract | Events

Latest 25 from a total of 720 transactions

Txn Hash	Method	Block	Age	From		To	Value	Txn Fee
0x1972d18fe1762d3415…	0x38f3d3fd	10386838	4 days 12 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000318252981
0xa0d9fdbd4673af31566…	0x38f3d3fd	10386147	4 days 15 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000181984413
0xe02e8ebcc415495e70…	0x38f3d3fd	10384902	4 days 19 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000181984413
0xe7afa6695e2754030d…	0x38f3d3fd	10384883	4 days 19 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000181984413
0x0ad604ea85f7c727e2…	0x38f3d3fd	10384840	4 days 19 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000181984413
0x093f6ee08c5ade8977…	0x38f3d3fd	10384641	4 days 20 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000181984413
0x49c94d4db1af51f92d3…	0x38f3d3fd	10384549	4 days 21 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x9fe64bf433721354404…	0 Ether	0.000181984413

Fig. 22. Moderna smart contract's view in Etherscan.



Contract 0x4B4bf83C0a46D22146428be5EEAcd8CcD35B1b17

Contract Overview

Balance: 0 Ether

More Info

My Name Tag: Not Available

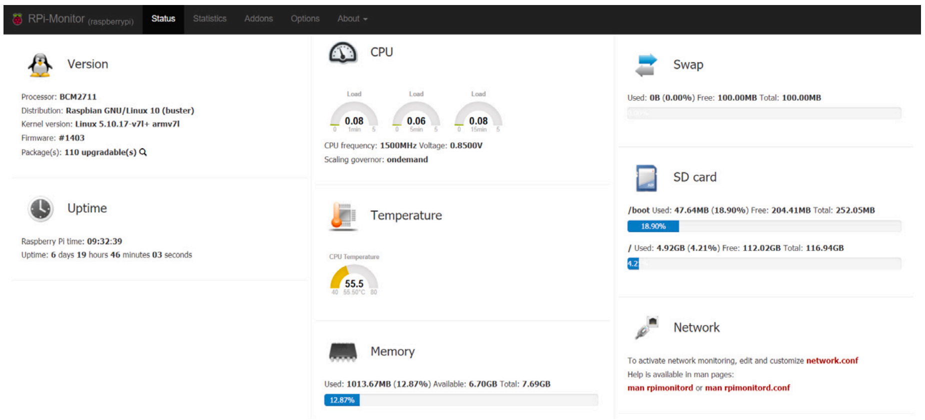
Creator: 0x0fa8dfe2e162e1e6f87f… at txn 0xba7901037db97aa219…

Transactions | Contract | Events

Latest 25 from a total of 501 transactions

Txn Hash	Method	Block	Age	From		To	Value	Txn Fee
0xf45b428f543478446e9…	0x38f3d3fd	10401714	2 days 2 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482
0x85108111bdc9bfe03d9…	0x38f3d3fd	10401601	2 days 2 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482
0x09ed1f258bbe34f217c…	0x38f3d3fd	10401533	2 days 2 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482
0xe252d3bb6fdfb249324…	0x38f3d3fd	10401524	2 days 2 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000203001279
0x9fe1796b8ad2b914a0…	0x38f3d3fd	10401388	2 days 3 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482
0xe9e2df9b674df3f4764…	0x38f3d3fd	10401365	2 days 3 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482
0xbae35d529215ccd2c1…	0x38f3d3fd	10401108	2 days 4 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482
0xf085e875d67afbe8efd…	0x38f3d3fd	10398708	2 days 10 hrs ago	0x0fa8dfe2e162e1e6f87f…	IN	0x4b4bf83c0a46d22146…	0 Ether	0.000104905482

Fig. 23. AstraZeneca smart contract's view in Etherscan.



Version

Processor: **BCM2711**
Distribution: **Raspbian GNU/Linux 10 (buster)**
Kernel version: **Linux 5.10.17-v7l+ armv7l**
Firmware: **#1403**
Package(s): **110 upgradable(s)**

Uptime

Raspberry Pi time: **09:32:39**
Uptime: **6** days **19** hours **46** minutes **03** seconds

CPU

Load 0.08 (1min), Load 0.06 (5min), Load 0.08 (15min)
CPU frequency: **1500MHz** Voltage: **0.8500V**
Scaling governor: **ondemand**

Temperature

CPU Temperature 55.5 (55.50°C)

Memory

Used: **1013.67MB (12.87%)** Available: **6.70GB** Total: **7.69GB**
12.87%

Swap

Used: **0B (0.00%)** Free: **100.00MB** Total: **100.00MB**

SD card

/boot Used: **47.64MB (18.90%)** Free: **204.41MB** Total: **252.05MB**
18.90%
/ Used: **4.92GB (4.21%)** Free: **112.02GB** Total: **116.94GB**

Network

To activate network monitoring, edit and customize **network.conf**
Help is available in man pages:
man rpimonitord or **man rpimonitord.conf**

Fig. 24. Summary of the RPi-Monitor main web interface.

well as statistics of some metrics about the performance of the low-cost device (see Fig. 24).

Fig. 25 illustrates the amount of computational work performed by our whole proposal running on a Raspberry Pi 4 device for one week. As we can see, the CPU consumption required by the proposal is low (between 0.0 and 0.2 min). Although there are some peaks, this is not an issue since their frequency is usually about once per day.

The status of the main memory of such a device is shown in Fig. 26. Although this device has a total memory of 8 GB (see Section 5), our proposal has not required so much capacity: the device could run the whole system without problems by using 1 GB approximately.

Fig. 27 presents the temperatures of the device while it was in execution. These values were usually stable although have suffered small variations. Note that these values were usually not higher than 55 °C and less than 45 °C.

Taking these results into account, we can conclude that EDALoCo can be appropriately deployed on lightweight devices. By using software containers, we have achieved that our approach is lightweight and can be executed on these devices without requiring a lot of resources. This validates our proposal as an efficient and feasible low-code approach to be deployed in this type of devices.

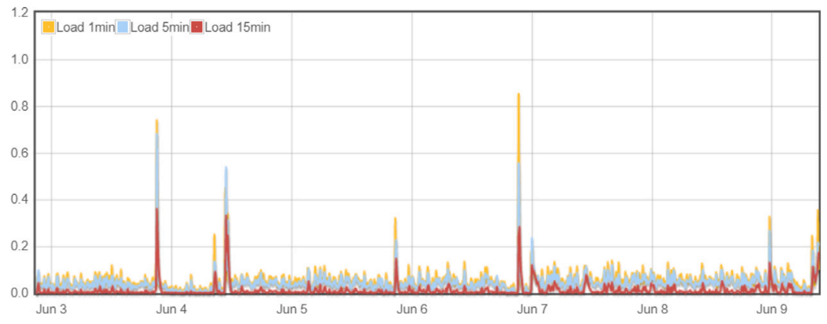


Fig. 25. Evolution of the CPU load on the Raspberry.

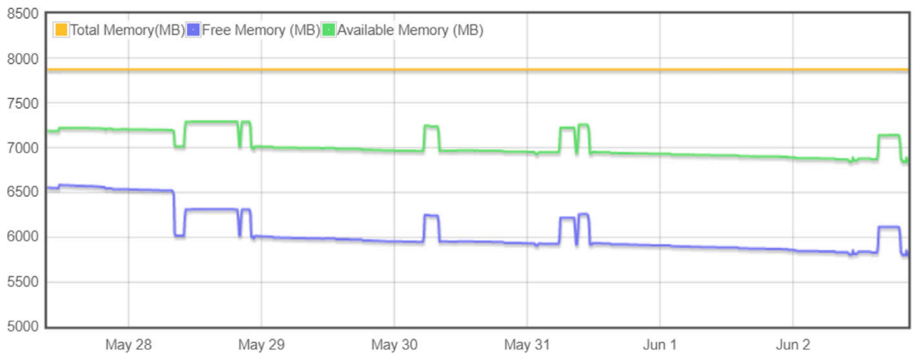


Fig. 26. Evolution of the main memory on the Raspberry.

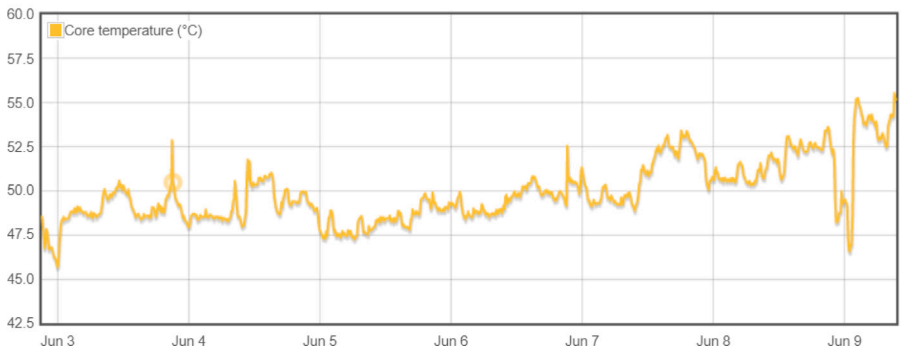


Fig. 27. Evolution of the temperature on the Raspberry.

6.3. *Threats to validity*

Next, we will discuss the threats to the validity of the evaluation results presented in this section. Specifically, we will divide them into four categories:

- Threats to internal validity, which consider factors that have not been controlled at the time of the development of the new nodes developed for the Node-RED tool and that may affect the results obtained.
- Threats to external validity consider whether the results of our case study can be generalized and adapted to other situations, i.e., whether we have achieved an adequate level of abstraction.
- Threats to construct validity in which we will check whether we are working with properties of interest.
- Threats to conclusions validity that will be those that put at risk our approach and the results obtained in this paper.

In terms of threats to internal validity, the main concerns we have taken into account are programming failures in the new nodes developed to extend the Node-RED tool. To mitigate the impact generated by this threat, we have decided to perform multiple tests on these nodes once implemented, using different examples and case studies with different situations to check their correct operation since we could manually check the results in the console of the low-code tool itself. To reach the adequate level of abstraction required by the low-code paradigm, it is important to ensure a good implementation that controls all possible situations and their exceptions. For this purpose, certain fields are mandatory without which the execution of the nodes will not be possible. In addition, an exception control has been implemented to control the different possible situations that may occur, showing the user what has happened in case something has not worked properly and what has been the reason.

With respect to external validity threats, the first thing we have considered is the ability of our approach to adapt to different domains. For our case study, we have used smart sensors connected to the low-code platform, which is hosted on a lightweight device. If our approach is able to provide good performance when applying it to this type of case study, then we can be confident that the performance will not be affected when applied to other types of domains. This case study is representative since we use different layers in our architecture, and the case study has a real representation in each of these layers. Therefore, the results obtained through these tests should be similar to the results that will be obtained by using this approach in other different domains. Finally, thanks to the use of the low-code paradigm we can abstract end users from technical details of the implementation of each of the nodes used [69].

Regarding the threats to the validity of the construct, we have taken into account the capability of the newly developed nodes with respect to the detection of errors derived from bad practices in their use. In order to do this, in our experiments we have introduced different anomalous situations not expected by the nodes to check what happens. Errors were always detected in a timely manner and displayed so that they could be corrected. Errors that are not detected or errors whose descriptions do not correctly describe the error are a threat to the validity of the construct. Furthermore, with respect to the performance evaluation of our proposal we have considered several metrics. In particular, Fig. 24 shows some of the metrics used, such as temperature, computational work and capacity of the main memory of the lightweight device.

Finally, the threats to the validity of the conclusions will be those factors that may compromise the conclusions we have reached based on the results. As mentioned above, thanks to the use of the low-code paradigm, we have achieved an adequate level of abstraction. This allows our approach to be used in other real-world case studies in addition to the one presented in this paper; however, the occurrence of any errors in some of the layers of our architecture (see Section 4) would be a threat to the validity of our conclusions. For instance, a malfunction of sensors or a blockchain saturation, i.e. when a blockchain network is not available or its performance is not as expected.

7. Conclusions and future work

In this paper, we proposed EDALoCo, an approach that enhances the accessibility of the blockchain technology by utilizing the low-code paradigm to develop event-driven applications for smart contract management.

EDALoCo, which is based on a containerized low-code platform, allows software developers who are non-experts in blockchain, to graphically develop event-driven applications to manage smart contracts in the blockchain network through a browser-based editor. This editor is the result of extending the Node-RED tool with novel blockchain-based nodes which we developed in this work. Thanks to this editor, such event-driven applications can be defined—even in a collaborative way—as graphical flows to be deployed in runtime through a single click.

The use of software containers in EDALoCo allows us to have a lightweight solution that can be run on most devices without worrying about the particular configuration or operating system supported by these devices. In addition, it offers us a modular solution in which we can deploy our proposal in one container and integrate it with other services. For example, the low-code platform in one container and a message broker in another one.

More specifically, EDALoCo was successfully applied to a case study on AstraZeneca's and Moderna's COVID-19 vaccines with the aim of real-time monitoring the temperatures to which these vaccines are exposed as well as automatically storing them in an Ethereum blockchain. That way, this immutable information is made public for anyone interested in verifying the cold chain. By using the proposed graphical editor, the user can model the event-driven applications necessary to manage the smart contracts in charge of controlling the logic of this case study.

As future work, we plan to apply EDALoCo to other real-world case studies, such as the monitoring of air quality or water consumption in smart cities [71], and e-health data [72,73]. We also intend to extend the palette of the low-code platform's editor to calculate statistics from blockchain data. This will allow users to detect situations of interest, such as blockchain network anomalies, or to analyze when it is cheaper (in terms of gas price) to register a temperature reading in a blockchain network.

CRediT authorship contribution statement

Jesús Rosa-Bilbao: Conceptualization, Methodology, Software, Validation, Investigation, Resources, Visualization, Writing – original draft, Writing – review & editing. **Juan Boubeta-Puig:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Writing – review & editing, Funding acquisition, Supervision. **Adrian Rutle:** Conceptualization, Methodology, Writing – review & editing.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.csi.2022.103676.

Acknowledgments

This work was supported by the Spanish Ministry of Science and Innovation and the European Regional Development Fund under projects FAME [RTI2018-093608-B-C33] and AwESOMe [PID2021-122215NB-C33], and the Research Plan from the University of Cadiz and Grupo Energético de Puerto Real S.A. under project GANGES [IRTP03_UCA].

References

[1] X. Xu, I. Weber, M. Staples, Architecture for Blockchain Applications, Springer International Publishing, Cham, 2019, http://dx.doi.org/10.1007/978-3-030-03035-3.

[2] Gartner, Blockchain technology & how it helps business growth, 2022, https://www.gartner.com/en/information-technology/insights/blockchain. (Accessed 03 February 2022).

[3] A. Azaria, A. Ekblaw, T. Vieira, A. Lippman, MedRec: Using blockchain for medical data access and permission management, in: 2nd International Conference on Open and Big Data (OBD), 2016, pp. 25–30, http://dx.doi.org/10.1109/OBD.2016.11.

[4] M. Liu, Z. Zhang, W. Chai, B. Wang, Privacy-preserving COVID-19 contact tracing solution based on blockchain, Comput. Stand. Interfaces 83 (2022) 103643, http://dx.doi.org/10.1016/j.csi.2022.103643.

[5] X. Zheng, Y. Zhao, H. Li, R. Chen, D. Zheng, Blockchain-based verifiable privacy-preserving data classification protocol for medical data, Comput. Stand. Interfaces 82 (2022) 103605, http://dx.doi.org/10.1016/j.csi.2021.103605.

[6] Y. Yu, Y. Zhao, Y. Li, X. Du, L. Wang, M. Guizani, Blockchain-based anonymous authentication with selective revocation for smart industrial applications, IEEE Trans. Ind. Inf. 16 (5) (2020) 3290–3300, http://dx.doi.org/10.1109/TII.2019.2944678.

[7] D. Puthal, N. Malik, S.P. Mohanty, E. Kougianos, C. Yang, The blockchain as a decentralized security framework [future directions], IEEE Consum. Electr. Mag. 7 (2) (2018) 18–21, http://dx.doi.org/10.1109/MCE.2017.2776459.

[8] D. Li, D. Han, Z. Zheng, T.-H. Weng, H. Li, H. Liu, A. Castiglione, K.-C. Li, MOOCsChain: A blockchain-based secure storage and sharing scheme for MOOCs learning, Comput. Stand. Interfaces 81 (2022) 103597, http://dx.doi.org/10.1016/j.csi.2021.103597.

[9] L. Tan, H. Xiao, K. Yu, M. Aloqaily, Y. Jararweh, A blockchain-empowered crowdsourcing system for 5G-enabled smart cities, Comput. Stand. Interfaces 76 (2021) 103517, http://dx.doi.org/10.1016/j.csi.2021.103517.

[10] D.S.W. Ting, L. Carin, V. Dzau, T.Y. Wong, Digital technology and COVID-19, Nat. Med. 26 (4) (2020) 459–461, http://dx.doi.org/10.1038/s41591-020-0824-5.

[11] J. Rosa-Bilbao, J. Boubeta-Puig, RectorDApp: Decentralized application for managing university rector elections, in: 2021 IEEE International Conference on Service-Oriented System Engineering, SOSE, IEEE, Oxford, United Kingdom, 2021, pp. 161–165, http://dx.doi.org/10.1109/SOSE52839.2021.00024.

[12] R.T. Geraldi, S. Reinehr, A. Malucelli, Software product line applied to the Internet of things: A systematic literature review, Inf. Softw. Technol. 124 (2020) 106293, http://dx.doi.org/10.1016/j.infsof.2020.106293.

[13] M. Xie, Y. Yu, R. Chen, H. Li, J. Wei, Q. Sun, Accountable outsourcing data storage atop blockchain, Comput. Stand. Interfaces 82 (2022) 103628, http://dx.doi.org/10.1016/j.csi.2022.103628.

[14] D.J. Yaga, P.M. Mell, N. Roby, K. Scarfone, Blockchain Technology Overview, NIST Pubs 8202, NIST, Gaithersburg, MD, 2018, pp. 1–66, http://dx.doi.org/10.6028/NIST.IR.8202.

[15] J. Boubeta-Puig, J. Rosa-Bilbao, J. Mendling, CEPchain: A graphical model-driven solution for integrating complex event processing and blockchain, Expert Syst. Appl. 184 (Article 115578) (2021) http://dx.doi.org/10.1016/j.eswa.2021.115578.

[16] M. Wohrer, U. Zdun, Domain specific language for smart contract development, in: 2020 IEEE International Conference on Blockchain and Cryptocurrency, ICBC, IEEE, Toronto, ON, Canada, 2020, pp. 1–9, http://dx.doi.org/10.1109/ICBC48266.2020.9169399.

[17] I. Grigg, The ricardian contract, in: Proceedings of the First IEEE International Workshop on Electronic Contracting, 2004, IEEE, San Diego, CA, USA, 2004, pp. 25–31, http://dx.doi.org/10.1109/WEC.2004.1319505.

[18] B. Jiang, Y. Liu, W.K. Chan, ContractFuzzer: Fuzzing smart contracts for vulnerability detection, in: Proceedings of the 33rd ACM/IEEE International Conference on Automated Software Engineering, in: ASE 2018, Association for Computing Machinery, New York, NY, USA, 2018, pp. 259–269, http://dx.doi.org/10.1145/3238147.3238177.

[19] L. Almonte, I. Cantador, E. Guerra, J. de Lara, Towards automating the construction of recommender systems for low-code development platforms, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, in: MODELS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–10, http://dx.doi.org/10.1145/3417990.3420200.

[20] OpenJ.S. Foundation, Node-RED, 2022, https://nodered.org/. (Accessed 03 February 2022).

[21] M. Wohrer, U. Zdun, Design patterns for smart contracts in the ethereum ecosystem, in: 2018 IEEE International Conference on Internet of Things (IThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom) and IEEE Smart Data (SmartData), IEEE, 2018, pp. 1513–1520, http://dx.doi.org/10.1109/Cybermatics_2018.2018.00255.

[22] G. Ortiz, J. Boubeta-Puig, J. Criado, D. Corral-Plaza, A.G. de Prado, I. Medina-Bulo, L. Iribarne, A microservice architecture for real-time IoT data processing: A reusable web of things approach for smart ports, Comput. Stand. Interfaces 81 (2022) 103604, http://dx.doi.org/10.1016/j.csi.2021.103604.

[23] Raspberry Pi, Raspberry pi, 2022, https://www.raspberrypi.org/. (Accessed 03 February 2022).

[24] W. Felter, A. Ferreira, R. Rajamony, J. Rubio, An updated performance comparison of virtual machines and linux containers, in: IEEE International Symposium on Performance Analysis of Systems and Software, ISPASS, 2015, pp. 171–172, http://dx.doi.org/10.1109/ISPASS.2015.7095802.

[25] C. Silva, J. Vieira, J.C. Campos, R. Couto, A.N. Ribeiro, Development and validation of a descriptive cognitive model for predicting usability issues in a low-code development platform, Hum. Factors 63 (6) (2021) 1012–1032, http://dx.doi.org/10.1177/0018720820920429.

[26] M. Ananthanarayanan, R. Mishra, V. Chakka, How integrated process management completes the blockchain jigsaw, Blockchain (2018) 4–7.

[27] D. Drescher, Blockchain Basics: A Non-Technical Introduction in 25 Steps, 1, A Press, 2017.

[28] K. Mammadzada, M. Iqbal, F. Milani, L. García-Bañuelos, R. Matuleviius, Blockchain oracles: A framework for blockchain-based applications, in: A. Asatiani, J.M. García, N. Helander, A. Jiménez-Ramírez, A. Koschmider, J. Mendling, G. Meroni, H.A. Reijers (Eds.), Business Process Management: Blockchain and Robotic Process Automation Forum, in: Lecture Notes in Business Information Processing, Springer International Publishing, Cham, 2020, pp. 19–34, http://dx.doi.org/10.1007/978-3-030-58779-6_2.

[29] Ethereum Foundation, Ethereum, 2022, Ethereum.Org https://ethereum.org. (Accessed 03 February 2022).

[30] A.J. Varela-Vaca, A.M.R. Quintero, Smart contract languages: A multivocal mapping study, ACM Comput. Surv. 54 (1) (2021) http://dx.doi.org/10.1145/3423166.

[31] Solidity, Solidity documentation, 2022, https://docs.soliditylang.org/en/v0.8.4/. (Accessed 03 February 2022).

[32] T. Chen, X. Li, X. Luo, X. Zhang, Under-optimized smart contracts devour your money, in: 2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering, SANER, 2017, pp. 442–446, http://dx.doi.org/10.1109/SANER.2017.7884650.

[33] C. Ploder, R. Bernsteiner, S. Schlögl, C. Gschliesser, The future use of Low-Code/NoCode platforms by knowledge workers – an acceptance study, in: L. Uden, I.-H. Ting, J.M. Corchado (Eds.), Knowledge Management in Organizations, Springer International Publishing, Cham, 2019, pp. 445–454.

[34] J. Caldeira, F. Brito e Abreu, J. Cardoso, J.P. dos Reis, Unveiling process insights from refactoring practices, Comput. Stand. Interfaces 81 (2022) 103587, http://dx.doi.org/10.1016/j.csi.2021.103587.

[35] L. Brunschwig, R. Campos-López, E. Guerra, J. de Lara, Towards domain-specific modelling environments based on augmented reality, in: IEEE/ACM 43rd International Conference on Software Engineering: New Ideas and Emerging Results, ICSE-NIER, 2021, pp. 56–60, http://dx.doi.org/10.1109/ICSE-NIER52604.2021.00020.

[36] A. Calderón, J. Boubeta-Puig, M. Ruiz, MEdit4CEP-Gam: A model-driven approach for user-friendly gamification design, monitoring and code generation in CEP-based systems, Inf. Softw. Technol. 95 (2018) 238–264, http://dx.doi.org/10.1016/j.infsof.2017.11.009.

[37] M. Brambilla, J. Cabot, M. Wimmer, Model-Driven Software Engineering in Practice, 2nd, Morgan & Claypool Publishers, 2017.

[38] J. Cabot, Low-code vs model-driven: are they the same?, 2022, https://modeling-languages.com/low-code-vs-model-driven/. (Accessed 03 February 2022).

[39] J. Cabot, Positioning of the low-code movement within the field of model-driven engineering, in: Proceedings of the 23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings, in: MODELS '20, Association for Computing Machinery, New York, NY, USA, 2020, pp. 1–3, http://dx.doi.org/10.1145/3417990.3420210.

[40] L.N. Sánchez-Morales, G. Alor-Hernández, V.Y. Rosales-Morales, C.A. Cortes-Camarillo, J.L. Sánchez-Cervantes, Generating educational mobile applications using UIDPs identified by artificial intelligence techniques, Comput. Stand. Interfaces 70 (2020) 103407, http://dx.doi.org/10.1016/j.csi.2019.103407.

[41] Mendix, The low-code guide, 2022, https://www.mendix.com/low-code-guide/. (Accessed 03 February 2022).

[42] S. Farshidi, S. Jansen, S. Fortuin, Model-driven development platform selection: four industry case studies, Softw. Syst. Model. (2021) http://dx.doi.org/10.1007/s10270-020-00855-w.

[43] B. Kenneweg, I. Kasam, M. McMullen, Building Low-Code Applications with Mendix: Discover Best Practices and Expert Techniques to Simplify Enterprise Web Development, vol. 20, Packt Publishing, 2021, pp. 1525–1551.

[44] B. Zarrin, H. Baumeister, Towards separation of concerns in flow-based programming, in: Companion Proceedings of the 14th International Conference on Modularity, in: MODULARITY Companion 2015, Association for Computing Machinery, New York, NY, USA, 2015, pp. 58–63, http://dx.doi.org/10.1145/2735386.2736752.

[45] F. Darema, Dynamic data driven applications systems: A new paradigm for application simulations and measurements, in: M. Bubak, G.D. van Albada, P.M.A. Sloot, J. Dongarra (Eds.), Computational Science - ICCS 2004, Springer Berlin Heidelberg, Berlin, Heidelberg, 2004, pp. 662–669.

[46] Mozilla, JavaScript, 2022, https://developer.mozilla.org/es/docs/Web/JavaScript. (Accessed 03 February 2022).

[47] H. Hasan, E. AlHadhrami, A. AlDhaheri, K. Salah, R. Jayaraman, Smart contract-based approach for efficient shipment management, Comput. Ind. Eng. 136 (2019) 149–159, http://dx.doi.org/10.1016/j.cie.2019.07.022.

[48] R. Singh, A.D. Dwivedi, G. Srivastava, Internet of things based blockchain for temperature monitoring and counterfeit pharmaceutical prevention, Sensors (Switzerland) 20 (14) (2020) 1–23, http://dx.doi.org/10.3390/s20143951.

[49] B. Yong, J. Shen, X. Liu, F. Li, H. Chen, Q. Zhou, An intelligent blockchain-based system for safe vaccine supply and supervision, Int. J. Inf. Manage. 52 (102024) (2020) http://dx.doi.org/10.1016/j.ijinfomgt.2019.10.009.

[50] M. Hamdaqa, L.A.P. Met, I. Qasse, iContractML 2.0: A domain-specific language for modeling and deploying smart contracts onto multiple blockchain platforms, Inf. Softw. Technol. 144 (2022) 106762, http://dx.doi.org/10.1016/j.infsof.2021.106762.

[51] Y. Li, G. Yang, W. Susilo, Y. Yu, M.H. Au, D. Liu, Traceable Monero: Anonymous cryptocurrency with enhanced accountability, IEEE Trans. Dependable Secure Comput. 18 (2) (2021) 679–691, http://dx.doi.org/10.1109/TDSC.2019.2910058.

[52] Y. Yu, Y. Li, J. Tian, J. Liu, Blockchain-based solutions to security and privacy issues in the Internet of things, IEEE Wirel. Commun. 25 (6) (2018) 12–18, http://dx.doi.org/10.1109/MWC.2017.1800116.

[53] Y. Li, W. Susilo, G. Yang, Y. Yu, X. Du, D. Liu, N. Guizani, Toward privacy and regulation in blockchain-based cryptocurrencies, IEEE Netw. 33 (5) (2019) 111–117, http://dx.doi.org/10.1109/MNET.2019.1800271.

[54] Y. Li, W. Susilo, G. Yang, Y. Yu, D. Liu, X. Du, M. Guizani, A blockchain-based self-tallying voting protocol in decentralized IoT, IEEE Trans. Dependable Secure Comput. 19 (1) (2022) 119–130, http://dx.doi.org/10.1109/TDSC.2020.2979856.

[55] G. Tian, Y. Hu, J. Wei, Z. Liu, X. Huang, X. Chen, W. Susilo, Blockchain-based secure deduplication and shared auditing in decentralized storage, IEEE Trans. Dependable Secure Comput. (2021) In press, http://dx.doi.org/10.1109/TDSC.2021.3114160.

[56] SettleMint, SettleMint, 2022, https://www.settlemint.com. (Accessed 03 February 2022).

[57] Unibright I.T. GmbH, Unibright, 2022, https://unibright.io. (Accessed 03 February 2022).

[58] Creator Platform, Creator, 2022, https://www.creatorchain.network. (Accessed 03 February 2022).

[59] Aurachain, Aurachain, 2022, https://aurachain.ch. (Accessed 03 February 2022).

[60] G. Ortiz, I. Castillo, A. Garcia-de Prado, J. Boubeta-Puig, Evaluating a flow-based programming approach as an alternative for developing CEP applications in IoT, IEEE Internet Things J. 9 (13) (2022) 11489–11499, http://dx.doi.org/10.1109/JIOT.2021.3130498.

[61] C. Silva, J. Vieira, J.C. Campos, R. Couto, A.N. Ribeiro, Development and validation of a descriptive cognitive model for predicting usability issues in a low-code development platform, Hum. Factors 63 (6) (2021) 1012–1032, http://dx.doi.org/10.1177/0018720820920429.

[62] COMET System, IoT wireless temperature datalogger, with built-in sensor, GSM modem and flat rate SIM card, 2022, https://bit.ly/3kfhedB. (Accessed 03 February 2022).

[63] International Organization for Standardization, ISO/IEC 17025 Testing and calibration laboratories, 2022, https://bit.ly/36INJc8. (Accessed 03 February 2022).

[64] Y. Quiñonez, C. Lizarraga, R. Aguayo, D. Arredondo, Communication architecture based on IoT technology to control and monitor pets feeding, J. UCS 27 (2) (2021) 190–207.

[65] European Medicines Agency, COVID-19 Vaccine AstraZeneca - product information, 2022, https://bit.ly/3orjQVC. (Accessed 03 February 2022).

[66] European Medicines Agency, COVID-19 vaccine moderna - product information, 2022.

[67] J. Rosa-Bilbao, J. Boubeta-Puig, A. Rutle, Dataset for EDALoCo: Enhancing the accessibility of blockchains through a low-code approach to the development of event-driven applications for smart contract management, 2022, Mendeley Data, v1, http://dx.doi.org/10.17632/drxw92dz4f.3. (Accessed 09 August 2022).

[68] Docker, Docker docs, 2022, https://docs.docker.com/. (Accessed 03 February 2022).

[69] C. Wohlin, P. Runeson, M. Höst, M.C. Ohlsson, B. Regnell, A. Wesslén, Experimentation in Software Engineering, Springer, 2012, http://dx.doi.org/10.1007/978-3-642-29044-2.

[70] X. Berger, RPi-monitor documentation, 2022, https://xavierberger.github.io/RPi-Monitor-docs/. (Accessed 03 February 2022).

[71] D. Corral-Plaza, G. Ortiz, I. Medina-Bulo, J. Boubeta-Puig, MEdit4CEP-SP: A model-driven solution to improve decision-making through user-friendly management and real-time processing of heterogeneous data streams, Knowl.-Based Syst. 213 (2021) 106682, http://dx.doi.org/10.1016/j.knosys.2020.106682.

[72] I. Calvo, M.G. Merayo, M. Núñez, A methodology to analyze heart data using fuzzy automata, J. Intell. Fuzzy Systems 37 (6) (2019) 7389–7399, http://dx.doi.org/10.3233/JIFS-179348.

[73] V. Valero, G. Díaz, J. Boubeta-Puig, H. Macià, E. Brazález, A compositional approach for complex event pattern modeling and transformation to colored Petri nets with black sequencing transitions, IEEE Trans. Softw. Eng. 48 (7) (2022) 2584–2605, http://dx.doi.org/10.1109/TSE.2021.3065584.

Jesús Rosa-Bilbao is a Ph.D. student in the UCASE Software Engineering Research Group at the University of Cadiz (UCA), Spain. He received his B.Sc. degree in Computer Science specialized in Software Engineering from UCA in 2019. In addition, he received his M.Sc. degree in Cybersecurity from UCA, his M.Sc. degree in Project Management from the European Business School in Barcelona (ENEB) and his M.Sc. degree in Business Administration from ENEB in 2020. He also received his M.Sc. degree in Big Data and Business Intelligence from ENEB in 2021. His research interests include complex event processing, event-driven service-oriented architecture, business process modeling, blockchain and cybersecurity.

Juan Boubeta-Puig received the Ph.D. degree in Computer Science and Engineering from the University of Cadiz (UCA), Cádiz, Spain, in 2014. He is an Associate Professor with the Department of Computer Science and Engineering, UCA. His research interests include real-time big data analytics through complex event processing, event-driven service-oriented architecture, Internet of things, blockchain and model-driven development of advanced user interfaces, and their application to smart cities, industry 4.0, e-health, and cybersecurity. Dr. Boubeta-Puig was honored with the Extraordinary Ph.D. Award from UCA and the Best Ph.D. Thesis Award from the Spanish Society of Software Engineering and Software Development Technologies.

Adrian Rutle holds a Ph.D. in Computer Science from the University of Bergen, Norway. Rutle is a professor at the Department of Computer science, Electrical engineering and Mathematical sciences at the Western Norway University of Applied Sciences (HVL), Bergen. Rutle's main interest is applying theoretical results from the field of model-driven software engineering to practical domains and has expertise in the development of modelling frameworks and domain-specific modelling languages. He also conducts research in the fields of modelling and simulation for robotics, eHealth, digital fabrication, smart systems and machine learning.