Future Generation Computer Systems 147 (2023) 219–234

Contents lists available at ScienceDirect

Future Generation Computer Systems

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

A Flexible Approach to Multi-party Business Process Execution on Blockchain

Flavio Corradini [a], Alessandro Marcelletti [a], Andrea Morichetta [a,*], Andrea Polini [a], Barbara Re [a], Francesco Tiezzi [b]

[a] University of Camerino, Camerino, Italy
[b] Università degli Studi di Firenze, Florence, Italy

ARTICLE INFO

Article history:
Received 11 November 2022
Received in revised form 28 April 2023
Accepted 5 May 2023
Available online 12 May 2023

Keywords:
Multi-party business process
BPMN
Blockchain
Smart contracts
Flexibility
Drools

ABSTRACT

In modern business scenarios, more and more organisations have to deal with the critical requirements of trustworthiness and flexibility, when collaborating in multi-party business processes. This calls for new kinds of systems able to manage collaborative processes in untrusted and dynamic environments. Concerning the collaborative perspective, the Business Process Management discipline has provided effective and standardised solutions for a long time, now. Regarding the trustworthiness perspective, blockchain is advocated as one of the most prominent technologies to guarantee trust in a multi-party setting. However, while the immutability of blockchain provides transparent and secure proof of past business interactions, it hinders the flexibility of the business process execution, as the business logic regulating the process execution is immutably stored in the blockchain. On the other hand, flexibility is a property that is becoming crucial in such a setting due to the high dynamism of the business scenarios. In fact, it permits to modify a process at run-time to deal with internal or external changes. In this paper, we face this issue by proposing an architecture for the flexible blockchain-based execution of multi-party business processes. In our approach, business processes are modelled by BPMN choreography diagrams translated into code, whose execution state is then stored in the blockchain. Flexibility is achieved by decoupling the business process's logic from its execution state, thus allowing run-time changes to the process execution without losing the fundamental properties of trust provided by the blockchain. To show the effectiveness of our approach, we provide a prototypical implementation, called FlexChain, and we use it on a case study from the healthcare application domain. The results obtained by the analysis of cost for the reported case study show the feasibility of the approach. In particular, major costs to sustain relate to one-time operations, such as the deployment and the run-time update of the model, while the most frequent actions are quite efficient.

1. Introduction

Nowadays, the success of enterprises increasingly depends on their capacity to interact with each other to achieve their goals collaboratively and to create new forms of business. To support these scenarios, the Business Process Management (BPM) discipline provided in the last two decades well-established, effective and standardised solutions, which come as a result of research activities and are largely adopted by the industry. In this respect, multi-party business processes can be considered one of the instruments permitting this technology revolution. The Business Process Modelling Notation (BPMN) [1] standard permits to represent business processes with different diagrams. Among them, the *choreography diagram* describes the interactions between multiple parties of a collaborative system in terms of the exchange of messages from a global perspective, without exposing the internal behaviour of the participants.

In collaborative systems, one of the most critical requirements is trustworthiness, given that such interactions have to occur in a distributed environment where trust cannot be typically assumed and is generally difficult to guarantee. In the BPM field, a well-established solution to this issue relies on the integration of blockchain technology, as also envisioned in [2–4]. The blockchain, indeed, guarantees the storage of data in an immutable and transparent form through a decentralised infrastructure. At its turn, this enables the auditing of all activities and data resulting from the process execution. In this respect, many technical solutions have been proposed [5–9]. They aim at regulating the interactions between parties using smart contracts

* Corresponding author.
E-mail addresses: flavio.corradini@unicam.it (F. Corradini), alessand.marcelletti@unicam.it (A. Marcelletti), andrea.morichetta@unicam.it (A. Morichetta), andrea.polini@unicam.it (A. Polini), barbara.re@unicam.it (B. Re), francesco.tiezzi@unifi.it (F. Tiezzi).

https://doi.org/10.1016/j.future.2023.05.006

(i.e., code running in the blockchain) automatically generated from BPMN diagrams.

However, while the immutable nature of blockchain permits to achieve trustworthiness by providing transparent and secure proof of past interactions, at the same time it hinders the flexibility of the business process execution [10]. Flexibility, in fact, is a crucial property in such a setting due to the high dynamism of the interactions in business scenarios permitting, e.g., an enterprise to be competitive in the market and succeeds in its business goals [11].

With the term flexibility, the BPM community indicates the need for business processes that are not static, and that instead embed mechanisms permitting to react to changes occurring at run-time. In particular, in this work, we refer to *flexibility by change*, which is the ability to modify a business process definition at run-time and migrate the current execution to the newly uploaded definition [11]. The flexibility property is a relevant activity for aligning business processes [12–14], and many solutions have been proposed for traditional (not based on blockchain) implementations, like those surveyed in [15]. However, flexibility is still an open challenge when blockchain technology is used to support the execution [2]. Flexibility is indeed a desirable property in such a context, and it could be required to deal with factors exogenous and endogenous to the blockchain-based business process (new laws, market dynamics or changes in customers' attitudes). An example of an exogenous cause is the Covid-19 pandemic, which forced in 2020 many private and public organisations to change their processes guaranteeing business continuity. Instead, an example of an endogenous cause is the change in the internal organisation model of one company involved in a partnership, which requires the companies to adapt their behaviours in the collaboration. In these situations, a blockchain-based implementation does not provide the possibility of updating the underlying running smart contracts, thus requiring the modelling and deployment of a new process. This brings additional costs in terms of time and money, and the loss of connection between the data already registered in the blockchain and the new smart contracts.

In this work, we propose a **flexible approach to the execution of multi-party business processes on blockchain**. In particular, we introduce an architecture based on a public permissionless blockchain, which is the most used technology for addressing trust in distributed and unsafe environments. Our prototypical implementation uses Ethereum, which provides an expressive language (Solidity) for writing smart contracts. The proposed approach exploits BPMN choreography models for the specification of multi-party business processes. A model is automatically translated in an *on-chain smart contract*, representing the current execution state, and a *rule-based program*, corresponding to the choreography's logic which is executed in an off-chain processor. This has several advantages. Firstly, we decouple the business logic of the choreography from its execution state to enable changes at run-time of the logic without interrupting the execution. Secondly, the off-chain execution of the choreography and the storage of logic in the InterPlanetary File System (IPFS) reduces the execution costs and overcomes the limited storage capacity of the blockchain [16,17], thus mitigating the scalability issues of public blockchains. Thirdly, the rule-based implementation allows achieving the modularisation of the code in terms of rules, each one corresponding to single business activity. Indeed, since the rules are decoupled from the contract, their execution and update are independent of other code, avoiding additional complexity and potential errors.

To show the feasibility and effectiveness of our approach, we provide a prototypical implementation, called FlexChain. We used FlexChain to tackle a case study dealing with a healthcare system for the booking of X-rays analysis.

The work is a revised and extended version of the one proposed in [18]. In particular, we have:

- revised and improved the methodology introducing a new on-chain smart contract structure; specifically, a factory contract is introduced to reduce the costs, and to improve the efficiency of the system;
- introduced a voting system for the contract update, directly managed on-chain;
- included an external repository to store rules, hence reducing costs;
- provided a new tool, implementing all the functionalities of the new methodology, designed with a more user-friendly interface integrating also the modeller.

The rest of the paper is organised as follows. Section 2 includes a general presentation of the BPMN choreography notation, blockchain, IPFS technology, and rule-based systems. Section 3 introduces the proposed approach of flexible execution, with a focus on the main phases, while Section 4 shows the translation approach for the rules generation and the structure of the smart contracts with their functionalities. Section 5 reports the implementation details of the framework by resorting to the case study. Section 6 reviews relevant related works. Finally, Section 7 concludes the paper by touching upon directions for future work.

2. Background notions

In this section we introduce the main concepts and technologies on which the FlexChain approach relies on.

2.1. BPMN choreography diagrams

The BPMN standard [1] permits the representation of multi-party business processes with different perspectives. In particular, we rely on **choreography** diagrams, as they allow to express the interactions among different parties without exposing their internal behaviour. Using choreographies is indeed possible to focus only on the interaction protocols specified in the corresponding business contracts. Moreover, compared to other kinds of notation, BPMN choreographies provide a more intuitive and high-level specification of business contexts. Let us consider, for example, the Petri Net notation. On the one hand, this notation has formally defined semantics that enables verification via many available tools (see, e.g., [19,20]). Moreover, over the years, various extensions (e.g., Coloured Petri Nets) were made available for using Petri Nets in different contexts, enriching their expressiveness. On the other hand, Petri Nets, and their variants, do not provide an adequate level of abstraction for a model-driven approach, especially in the business domain. Indeed, Petri Nets may be too technical to be used by users not familiar with formalisms, and multi-party business scenarios may lead to Petri Net models being too complex and difficult to understand by business experts. Instead, BPMN choreographies permit the development of multi-party systems making the approach accessible, to some extent, also to non-technical users.

In this way, in a distributed environment, parties can refer to specific choreographies for collaborating, describing how the interaction should happen so as to achieve common objectives. This results in a peer-to-peer collaboration, in which each individual node has to take responsibility for its execution steps. Consequently, in a choreography approach, each participant is responsible for partial orchestration, based on its rules and without relying on a central coordinator, with final behaviour being expressed as a family of permitted message exchange sequences.

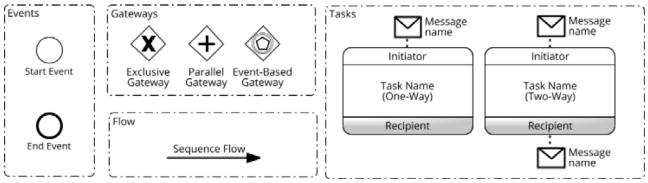


Fig. 1. BPMN choreography elements.

Fig. 1 shows the most relevant elements used in choreography diagrams. The business process's control flows are shown on the left side, while communication elements are on the right one. The main elements composing a choreography model are generally (i) events, (ii) gateways, (iii) sequence flows, and (iv) tasks. *Start* and *End* events represent the starting and ending points of the choreography. *Gateways* act as either split nodes (forking into outgoing edges) or join nodes (merging incoming edges). A *parallel gateway* (AND) in the split case initiates all the outgoing edges simultaneously, while in the join mode, the gateway has to wait to be reached by all its incoming edges to be activated. An *exclusive gateway* (XOR) represents choices; it activates exactly one outgoing edge in split mode and it acts as a pass-through in join mode. In the *event-based gateway*, the outgoing branch activation depends on the reception of a message; the message events connected to the gateway are in race condition: the first one that is triggered activates the corresponding branch and disables the other ones. *Sequence Flows* are edges used to specify the execution flow by connecting choreography elements. Finally, *Tasks* allows defining the message exchanges between the parties involved in the choreography. They are represented as rectangles divided into three bands: the central one contains the task's name, while the others refer to the involved parties (the white band is the initiator, and the grey one is the recipient). Messages can be sent either by one party (One-Way tasks) or by both parties (Two-Way tasks).

2.2. Blockchain

A blockchain is a distributed ledger composed of a set of transactions organised in blocks that are linked together, forming a chain structure, by the use of cryptography-based mechanisms. A distributed ledger is an append-only data structure, where each new block is added to the tail of the chain at regular intervals of time. The process of generating new blocks is the result of the *miners* work that are computational nodes composing the blockchain network. Thanks to this mining process and the use of a consensus protocol, the blockchain guarantees trustworthiness and decentralisation, without the need of introducing a third-party trusted authority. Initially, the blockchain was mainly used for payment purposes due to its adoption to create and manage cryptocurrencies, but in the last years, with the introduction of smart contracts, the application scenarios have drastically increased. From healthcare [21] to IoT [22] domains, blockchain is now creating opportunities for innovative applications bringing new research challenges [23]. These contracts can be considered special software programs, written by developers, that are installed (i.e., deployed) and executed over the blockchain infrastructure. The main characteristic of smart contracts relies on the immutability of code, as well as on their results that are stored like public transactions in the blockchain, making such information available for auditing purposes. Each blockchain technology generally provides a native language for specifying smart contracts. Such languages somehow reflect the characteristics and the mechanisms embedded in the corresponding blockchain. In our case, we rely on *Solidity*[1] for writing smart contracts for Ethereum. When a smart contract is created and executed inside the Ethereum blockchain, it is possible to program the so-called *events*. In certain parts of the Solidity code is indeed possible to program the emitting of those events that can contain different types of data. The result is the writing of this information in the events part inside the transaction log allowing external software to capture it.

The execution of smart contract operations, like other blockchain transactions, in Ethereum, has an associated cost measured in terms of *gas*. The total fee that the user has to pay is calculated by multiplying the amount of used gas with the *gas price* that defines the amount of Ether (i.e., the Ethereum cryptocurrency) to be paid for each unit of gas. This price can be specified by the user and a higher value guarantees a faster inclusion of the transaction in the blockchain. These execution fees encourage mining activities by network participants and, hence, permit to keep the overall system to work. Indeed, for each block mined, nodes are rewarded with a default amount of Ether plus the sum of the transaction fees included in the block. Notice, each user account in the Ethereum network is identified by a unique hex address that we exploited in our approach for identifying participants during the execution of the choreography instances. From a technical point of view, blockchain provides a direct communication environment between business participants, without the need for a trusted third party, or a central authority. Furthermore, the secure consensus algorithm and the transparency of a public blockchain provide participants with the guarantee of non-repudiable and auditable information. The aforementioned characteristics of the public blockchain make it the most recommended solution for the trustworthy execution of business processes in trustless contexts. Among the most known public blockchain implementations, we selected Ethereum since is one of the most stable and used technology. It is worth mentioning that currently, this technology results to be rather demanding in relation to requirements connected to time performance and financial costs so its adoption should be carefully assessed depending on each specific scenario. Anyway, being the Ethereum Virtual Machine (EVM) used by newly emerging solutions addressing those lacks, the choice of Ethereum allows high portability among the newly proposed technologies according to different needs.

2.3. Drools

Rule-based systems [24,25] are systems that manipulate data through *condition-action* rules expressed in a declarative way. They are evaluated by a rules engine that automatically executes the triggered actions.

In this work, we rely on the Drools[2] system, which is made up of the following components. The *Knowledge Base* represents the rules to evaluate. The *Database* stores the data matched by the conditions of the rules. The *Rules Engine* is the component that processes the knowledge base and executes the triggered actions. The *User interface* is the component that allows the users to interact with the engine and the knowledge. The execution of a rule-based program consists of the evaluation of the knowledge base rules against the data stored in the database by the rule engine. If a condition is satisfied, the engine executes the corresponding actions producing new results. One of the main advantages of these systems is that they allow the automation of many actions defined by domain experts. Indeed, the use of rules makes transparent the execution, thanks also to the structure of the rules that are easy to understand due to their human-oriented form.

[1] https://solidity.readthedocs.io/.

[2] https://www.drools.org/.

In our work, we use Drools for representing and executing the business logic, thanks to the specific characteristics provided by this rule engine. In particular, the main advantage of Drools, and that has mainly driven our choice, is the flexibility of this technology, which permits decoupling the business logic from the application code, with high benefits in terms of code maintenance and adaptation to changes. Other advantages are the understandability of rules, the separation of responsibilities and the reusability of the produced code [26].

2.4. IPFS

The InterPlanetary File System (IPFS) [27] is a distributed system for storing different types of data. It is based on a peer-to-peer network where each peer stores some information accessible from anywhere. Through the hashing of the information, IPFS binds data with a unique Content Identifier (CID) formed by the hash of the content itself. It uses the Interplanetary Linked Data (IPLD) ecosystem for formatting and storing data across the network. The used data structure corresponds to Merkle Directed Acyclic Graphs where each piece of information stored through the IPFS is split inside blocks. This makes the data versioning more efficient since it allows to update only part of the content, by uploading the new data and receiving back the new CID. In the querying phase, IPFS uses the Distributed Hash Table (DHT), which is a table that uses a key–value format to store the information and is distributed over the network. Using the libp2p project[3] is then possible to query those peers and those DHTs in order to retrieve the needed information.

3. The FlexChain approach

Our approach aims at guaranteeing the flexible execution of smart contracts following a model-driven technique. Starting from BPMN choreography diagrams, representing the business interactions between parties, the FlexChain framework generates an infrastructure based on a public blockchain. This guarantees at the same time a high level of trust and the flexibility useful to enable changes at run-time.

3.1. Architecture

The FlexChain's architecture is reported in Fig. 2. It is based on the pattern proposed in [28] and is divided into four main components: (i) Frontend component (ii) Blockchain, (iii) Off-chain processor, and (iv) IPFS.

The **Frontend component** is a sort of fat client providing functionalities to interact with the user (User interface) and to provide all the functionalities used for managing a choreography model and interacting with the blockchain. In particular, the Core sub-component exposes the model design, choreography instantiation, update and execution of a choreography instance. Thanks to this component, and our automatic translation approach, the end user does not need any technical knowledge about blockchain and its implementation.

The **blockchain** component is composed of two main smart contracts: the *choreography instance smart contract* and the *factory contract*. The former has a generic structure and is deployed automatically by the factory contract, which instead facilitates the managing of instances and acts as a monitor of each generated contract instance. The choreography instance smart contract is responsible for storing the current execution state of a choreography instance. This guarantees a trusted and reliable exchange of information among parties during the choreography execution.

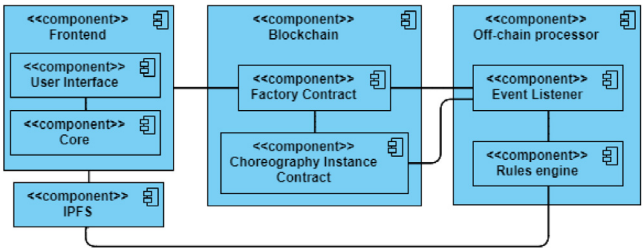


Fig. 2. Component diagram of the FlexChain's architecture.

Furthermore, the contract provides an immutable track of the execution history and all the choreography instance updates.

In such architecture, flexibility is achieved by implementing the execution logic of the choreography using Drools rules. The rules are generated automatically by the FlexChain framework and they have been deployed on the **IPFS** repository. The resulting hash is successively stored inside the on-chain choreography instance smart contract by the Frontend component[4] to have a certified reference of the rules during the execution. The benefit of using a rule-based technique comes from the possibility of having a direct representation of a choreography message as a rule that is executed independently from the others. This, combined with the off-chain storage, allows overcoming the restrictions of the blockchain, since in this way it is possible to update at run-time only the targeted rules, without being limited by the immutability of smart contracts. The other advantage of storing rules inside the IPFS regards the costs. Indeed, with this approach, for each update in the choreography, the smart contract has to change only the reference to the final IPFS hash. This reduces costs with respect to storing the whole set of rules inside the smart contract on-chain.

The **off-chain processor** is used to execute rules. Its behaviour is triggered directly by an event emitted by the choreography instance smart contract and caught by the *event listener* component. This component exploits the Ethereum API and waits for every event emitted by the smart contracts. In particular, its main role is to listen for message execution events, invoking then the Rules engine component by forwarding the necessary data. The event contains the necessary information regarding the rule to execute on the *rules engine* and the relative parameters. The off-chain processor is a component outside the control of the participants, indeed, only the smart contract can trigger the execution of tasks. Furthermore, the operations performed off-chain are auditable by all the participants since the operations are registered in the smart contract before and after their execution. To notice, at each execution the off-chain processor will always check the latest version of the rule present in the IPFS according to the registered id stored in the choreography smart contract. The use of an off-chain practice is nowadays widely common [29,30], in particular when combined with the blockchain to address storage and performance limitations. In these situations, the trust the blockchain provides derives from the capability to possibly run auditing sessions on the resulting execution traces. Indeed, in our case the smart contracts trigger the execution of the off-chain rules, receiving back the final status. All this information is stored on the blockchain, thus making it visible and verifiable by every participant. In this way, we can provide a balanced approach in terms of performance and trustworthiness.

To facilitate user interactions, in the Flexchain tool we also provide a graphical user interface that takes in input information

[3] https://libp2p.io/.

[4] For this reason, in Fig. 2 there is no direct connection between IPFS and Blockchain.

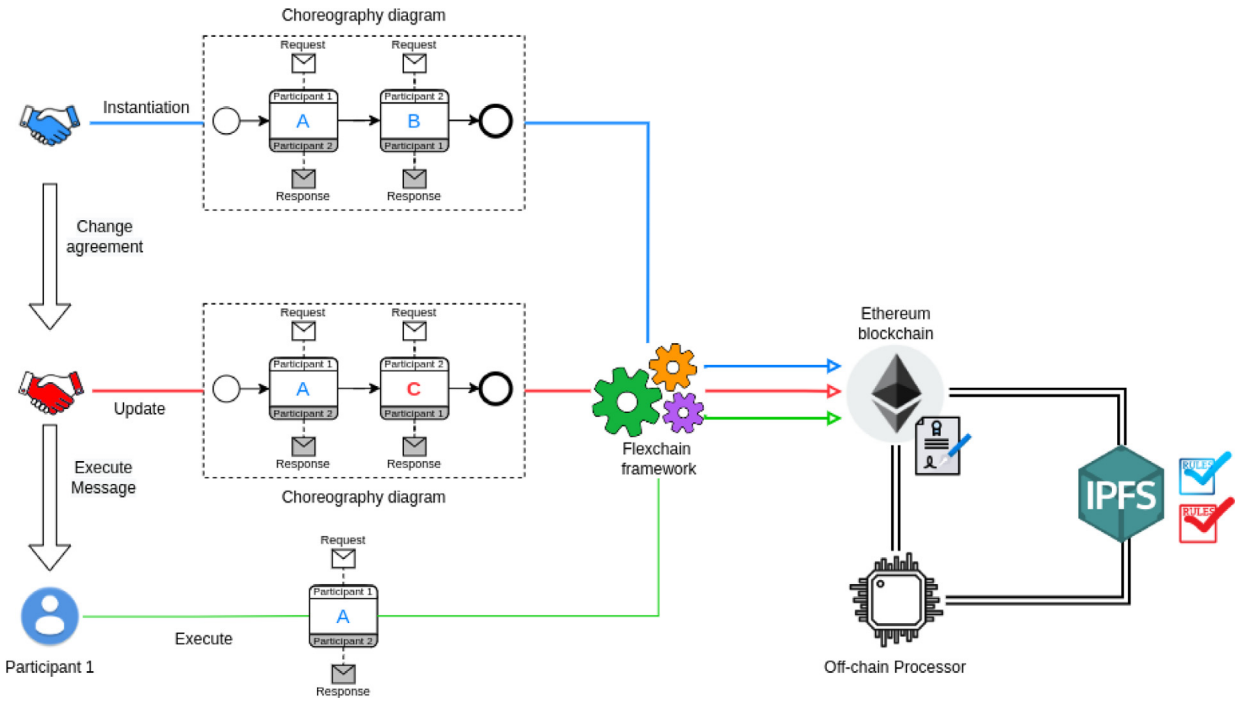


Fig. 3. The FlexChain approach.

from the end user and forwards them to the smart contract. From there, the smart contract and the off-chain processor automatically carry out all subsequent steps connected to the message reception. This kind of interaction is, indeed, the one considered in the case study in Section 5, where the participants of the choreography are humans. However, the framework can also support a fully-automatic form of interaction (via, e.g., REST services). In this case, the Flexchain user interface can be bypassed, and messages can be automatically executed by connecting an external application to the smart contracts.

3.2. Phases

The FlexChain approach is composed of three main phases (Fig. 3): the instantiation, the update and the execution of the activities. The **instantiation** is the first phase and it starts with the user uploading a choreography model using a dedicated interface. This model is then automatically translated into a set of rules and produces an on-chain smart contract instance. This contract is deployed through the use of the factory contract and represents the choreography instance state. The rules instead are stored inside the IPFS and they represent the execution logic of the process. In particular, they specify how a message can be sent, under which conditions and results the execution produces. Rules are automatically derived by a translation performed on the messages of the model and they are executed by an off-chain processor. The decoupling of the state from its logic is an important aspect of the proposed approach; it allows the use of the immutability of the blockchain for storing business information, while the logic and its constraints can be maintained off-chain. Thanks to this solution, we are able to achieve flexibility while keeping the immutability property of blockchain technology.

The **update** phase is similar to the previous one and it consists of the willingness of a participant to perform changes to a running process instance. In particular, in the FlexChain approach, changes are isolated and affect only the running instance to which they are applied. Indeed, we do not want to assume that the motivation behind a run-time change is valid for all the current active instances, as they may involve different participants with different needs and contexts. It is worth noting that the update of the rules does not affect the current instance state stored on-chain, permitting the new rules to continue operating in the current state. The willingness for a change in the process instance is expressed with the proposal of an updated model. The FlexChain system this time will produce neither a new contract nor a new instance version, but only a new set of translated rules derived from the updated model. Before being part of the effective execution, these rules are stored in a new IPFS location and the hash is stored on-chain in the running smart contract instance. At this point, the participants can use the instance contract for voting on the update. In case the quorum[5] is reached, the smart contract automatically replaces the old hash with the one just voted. In this way, we do not need to modify directly the smart contract code, since it contains only the current state of the process.

The last phase regards the **execution** of a message and the consequent state update. In this case, to start the execution, the user needs to select through the frontend interface the message to send to the counterpart inserting also the required parameters. The frontend then invokes the smart contract instance passing this information. The contract will successively emit an event that the off-chain processor will catch about the operation to perform. The processor retrieves the corresponding rule from the IPFS and, after executing it, will return the result to the smart contract.

3.3. FlexChain security aspects

It is worth mentioning that most of the security properties connected to the FlexChain approach are inherited from the underlying blockchain technology. The literature is plenty of works focusing on the security aspects of blockchain technologies [31]. Once assumed the security aspects of the blockchain, we need to discuss how the peculiarities of our approach could introduce possible threats. To this aim, the main aspects to consider are related to the potential malicious behaviour of participants during

[5] The quorum refers to the minimum number of choreography participants that need to accept and commit an updated proposal.

the update and/or execution phases. In the former case, a participant could propose a malicious update of the choreography without the consensus of the counterparts. Here, thanks to the use of an on-chain voting mechanism, all the participants need to vote on the proposal previously stored in the smart contract. Then, only in case the quorum is reached the smart contract will adopt the new rules. The assumption is that every party involved in the choreography is interested in continuing the correct execution and that they vote fairly during an update; otherwise, the proposal can be rejected, thus interrupting the update procedure. In the latter case, the introduction of an off-chain engine opens to new possible attacks leading to corrupted execution results. However, the use of blockchain makes it possible for the involved parties to monitor the current execution [32] and to assess whenever something does not abide by the initial agreement defined in the choreography. This could result in an interruption of the collaboration and the consequent execution.

4. Translation approach: rules and smart contracts

In this section, we provide the details for the generation of the rules and the smart contracts starting from choreography messages and control flow elements supported by BPMN. We report here the different translations according to the different elements and their connections.

4.1. Approach for rules generation

Our approach is based on a direct translation that starts from a BPMN choreography model and generates Drools rules. The input of the translation is a choreography model compliant with the BPMN standard, and that can include anyone of the BPMN elements reported in Fig. 1, which correspond to the most used elements in BPMN choreography diagrams [33]. Hence, no further additions to the input language are required for dealing with common application scenarios. If a new element is needed, the algorithm should be modified to include it. In fact, our work aims to demonstrate the feasibility of the approach, trying to optimise the code resulting from the translation as much as possible without the need to provide an extensible framework.

Messages are the only elements requiring direct enforcement and regulation. In fact, during the execution of a choreography, only the messages can actively modify the current state due to the participants' actions. Other directly connected elements, like gateways and events, are translated as conditions to be verified internally to messages. In this way, we are able to check whenever a decision derived by a gateway must be taken. For example, in an exclusive gateway, a boolean expression allows the rule to evaluate the right path to be followed. Also, to enforce the right execution sequence, an execution status is attached to each message and its related rule. Indeed, a message can be *enabled*, *disabled*, or *completed*; thus, depending on the status, only a certain action can be performed. For example, if a message is completed it cannot be executed again, and its corresponding rule will always fail if invoked. Notice that, in case of a loop in the model, the same message can be re-activated many times, in order to re-execute it.

To sum up, the translation of the choreography model encodes in the generated rules the model's control-flow, thus providing the choreography execution with an enforcing mechanism. In particular, to prevent unexpected execution, each rule checks the status of the involved elements according to the state variables stored in the smart contract. Moreover, the data exchanged during the execution is stored in the blockchain and can be audited by all the participants.

Translation algorithm. The FlexChain translator generates rules from a BPMN choreography diagram according to the following translation algorithm. The algorithm iterates over the list of all choreography messages and, for each of them, a new rule is created. For rule generation, the algorithm considers the paths leading to the corresponding message. Indeed, for each element in these paths, a condition is added inside the rule. If the element is an exclusive gateway, the related conditions are extracted from the sequence flow of the element, and added to the rule. If instead the element is a message, a check on the completion of this message is added inside the rule and the exploration of that path stops. Finally, after all the incoming paths connected to the considered message are explored, the code for pushing state variables to the smart contract state is added inside the 'Then' part of the rule.

4.2. Examples of translation results

To clarify how the translation algorithm described in Section 4.1 operates, we report here some translation examples resulting from applying the algorithm to typical excerpts of BPMN choreography models.

For each element, we show the corresponding pseudo-code of the rule and we discuss then the main peculiarities. As previously mentioned, the approach directly translates a BPMN choreography message into a Drools rule while the start and end events are not considered. Indeed, they only define the starting and ending points of the choreography without actively influencing the execution. For this reason, we transfer these behaviours to the initial and ending messages, without explicitly considering the events.

One-way task. The simplest element is represented by the one-way task shown in Fig. 4. Its translation generates the rule reported in Listing 1. The general formatting is composed of the rule name (Line 1), corresponding to a unique message-id, the *When* part (Line 2), containing the conditions to evaluate, and the *Then* part (Line 6), defining the actions to perform. Inside the *When* clause, different types of conditions are verified. Firstly, the statuses of the previous connected messages are checked (Line 3). Then, the rule checks the current element status, which must be enabled (Line 4). Line 5 reports the constraints derived from the guards of the exclusive gateways. Finally, in the *Then* clause, the state variable *var1* is pushed to the smart contract (Line 7) updating the contract state. This variable corresponds to the argument of the message, whose value is set by the user when the message is sent.

Listing 1: Rule of a one-way task.

```
1  Rule "message1"
2  When
3     <conditions on previous messages> &&
4     message1 == enabled &&
5     <conditions from exclusive gateways>
6  Then
7      push var1
8  End
```

It is worth noticing that, after the rule action (in the *Then* clause) is executed, the message status is automatically set to completed in the smart contract.

Sequence flow. Sequence flow connectors are used to specify the execution order of activities in a choreography. To exemplify how sequence flows affect the conditions of a rule, we consider the simple case in Fig. 5, where two one-way tasks are connected. The generated rule for message *message1* is as in Listing 1, while the rule for message *message2* is reported in Listing 2. In the *When*

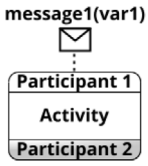


Fig. 4. One-way task.

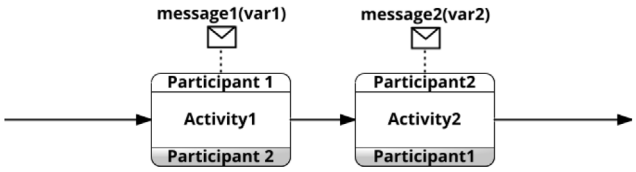


Fig. 5. Sequence flow.

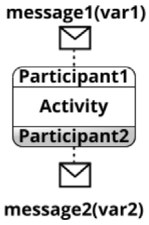


Fig. 6. Two-way task.

clause, the rule contains two conditions (Lines 11–12), expressing that the state of the previous message *message1* must be *completed* and the current state of *message2* must be *enabled*. This check is really important to enforce the execution of messages in the right sequence. If both conditions are satisfied, the *var2* variable is written in the blockchain (Line 14).

Listing 2: Rule of a task following another one.

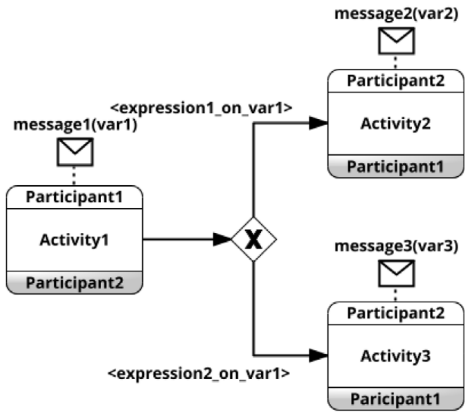
```
9   Rule "message2"
10  When
11      message1 == completed &&
12      message2 == enabled
13  Then
14      push var2
15  End
```

Two-way task. The two-way task, depicted in Fig. 6, represents two one-way tasks in sequence. Therefore, it is translated as two rules, one for each message, as in Listings 1 and 2.

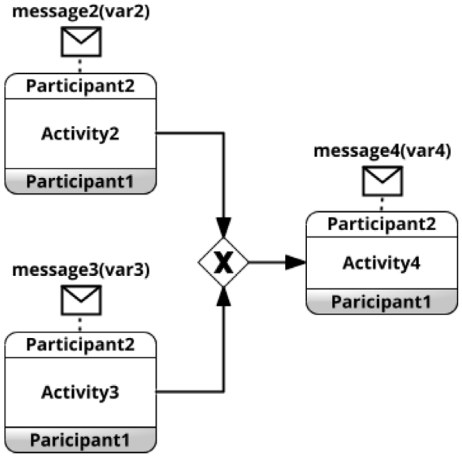
Gateways. We describe now how gateways affect the conditions of a rule when a message is directly connected to them. As the first case (reported in Fig. 7a) we consider a split exclusive gateway in which outgoing sequence flows contain boolean expressions on the *var1* variable. In Listing 3 we report the gateway translation example referring to the connected *message2*. The rule contains an additional boolean expression referring to the condition of the exclusive gateway (Line 20). Depending on the type, the compare operator inside the expression can be on integer, string or boolean values.

Listing 3: Rule of a message after a split exclusive gateway.

```
16  Rule "message2"
17  When
18      message1 == completed &&
19      message2 == enabled &&
20      expression1_on_var1 == true
21  Then
22      push var2
23  End
```



(a) Split case



(b) Join case

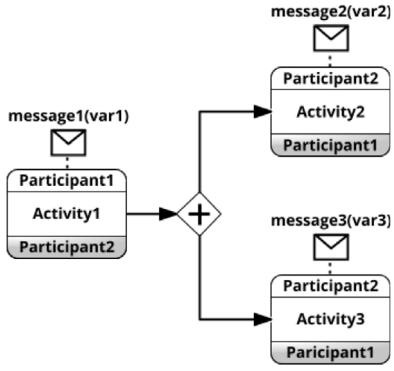
Fig. 7. Exclusive gateway elements.

Another case considering the exclusive gateway is the join one (Fig. 7b). This time there is no expression to verify, therefore the rule of *message4*, reported in Listing 4, contains only the execution status to control before the execution. Indeed *message4* requires that only one between *message2* and *message3* is completed (Lines 26–27).

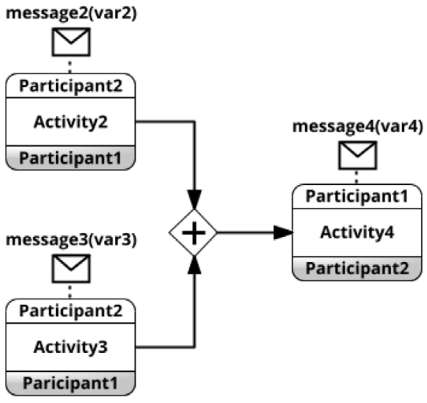
Listing 4: Rule of a message after a join exclusive gateway.

```
24  Rule "message4"
25  When
26      (message2 == completed
27        || message3 == completed) &&
28      message4 == enabled
29  Then
30      push var4
31  End
```

The next case regards the parallel gateway element that is divided again into a split and a join. The split gateway (Fig. 8a), enables the execution of all successive messages without any restriction. This behaviour does not affect directly any rule, neither of the messages before the gateway nor the ones after. All the messages on the right-hand side of the gateway are enabled after the completion of the previous one. For this reason, in Listing 5 the rule of *message2* does not contain any particular condition but it works as two directly connected messages.



(a) Split case



(b) Join case

Fig. 8. Parallel gateway elements.

Listing 5: Rule of the message after the split parallel gateway.

```
32  Rule "message2"
33  When
34      message1 == completed &&
35      message2 == enabled
36  Then
37      push var2
38  End
```

Different is the case for the parallel join (Fig. 8b). In this case, the *message4* requires that all the previous elements are completed before its execution. This semantics affects the initial check of the rule associated with *message4*. Specifically, Listing 6 shows (Lines 41–42) the parallel check in the condition for the execution status of the previous messages *message2* and *message3*, that need to be both completed.

Listing 6: Rule of a message after the join parallel gateway.

```
39  Rule "message4"
40  When
41      (message2 == completed
42       && message3 == completed) &&
43      message4 == enabled
44  Then
45      push var4
46  End
```

The last case reported in Fig. 9 shows an event-based gateway. The gateway as shown in Listing 7 requires that *message2* is enabled and, at the same time, *message3* is disabled, since only one message can be executed according to the one that is first chosen. This is reflected in the inclusion of an extra condition requiring that the status of the other message connected to the gateway is disabled. Lines 49–50 report the example for *message2* in which the check is done also on the status of *message3*.

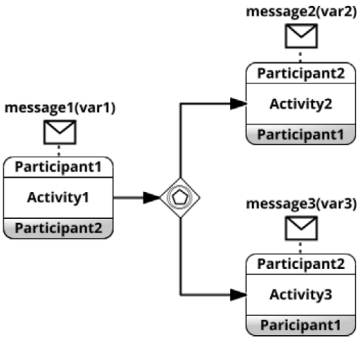


Fig. 9. Event-based case.

Listing 7: Rule of a message after event-based gateway.

```
47  Rule "message2"
48  When
49      (message1 == completed
50       && message3 == disabled) &&
51      message2 == enabled
52  Then
53      push var2
54  End
```

4.3. Smart contract

The FlexChain approach relies on two different smart contracts regulating the on-chain behaviour. The first is the *Factory contract* that is invoked by the FlexChain tool every time a choreography model is uploaded and a new choreography instance must be created. This acts also as a register since it maintains in memory the name of the choreography instance and its related address. The second is the *Instance contract* and it is used as a template for instantiating a model. The code is composed of a set of state variables, that are used to manage the process, and some utility functions for the different operations. In particular, in FlexChain we have a single Factory contract that deploys an Instance contract for each model instance generated by the user. This architecture provides a general and reusable infrastructure dealing with flexibility requirements. Notably, the translation examples shown in Section 4.2 do not produce smart contracts code, since the Drools rules generated from the BPMN elements are stored inside the IPFS and just referred to in the Instance contract.

Factory contract. Listing 8 reports the main functionalities for instantiating new contract instances. Line 56 contains the mapping used to associate the instance name (in bytes) to its address. In this way, it is possible to track the entire history of generated choreography contract instances. The *instantiateProcess* function (Lines 57–63) generates a new contract starting from the template one. The new contract requires two inputs, (i) the creator, automatically taken from the sender of the invocation, and (ii) the quorum, required for an update. After this step, the newly generated address is stored in the state mapping.

Listing 8: Factory Smart Contract.

```
55  contract ProcessMonitor{
56   mapping(bytes32 => address) processes;
57   function instantiateProcess(
```

```
58      bytes32 processName,
59      uint _quorum) public{
60          processes[processName] =
61          address(new ProcessTemplate(
62              msg.sender, _quorum));
63   }
64   ...
65  }
```

Instance contract. Listing 9 reports the code for the smart contract representing the choreography instance. In particular, in Line 67 the *enum* variable for managing the execution status of the rules is defined. Then, the hash location of the rules inside the IPFS is stored in a separate bytes32 variable (Line 68). The next mappings are used instead to associate the messages to an execution status (Line 69) and to store the input variables of the messages with their name (Line 70). Finally, two structures are used to manage the voting system for updating the process. The first (Line 71) represents the participants that vote for an update, while the second (Lines 72–76) handles the proposal. This contains indeed the new hash of the updated rules, the required quorum, and the actual votes received for that proposal. The next part of the contract defines the execution of the process instance. In Line 77 the *executeMessage* function is used to emit the event corresponding to the *messageToExecute* and its *inputs*. In the FlexChain approach, the above event is used to trigger the execution of the message rule; after this step, the result of the off-chain computation is pushed to the contract using the *setVariables* function (Line 82). Here the *names* and the *values* of the output variables are updated in the contract state and the message is set to the *COMPLETED* state. The function defined in Line 88 manages the creation of an update proposal, requiring the hash of the new rules location and the IDs of the corresponding messages. The last function in Line 93 permits to vote for the update proposal and it requires only the *vote* that will be associated automatically with the sender of the transaction (i.e., the choreography participant). The remaining part of the contract defines all the getter functions for reading the contract state and other information.

Listing 9: Choreography Instance Smart Contract.

```
66  contract ProcessTemplate{
67   enum State { ENABLED, DISABLED, COMPLETED }
68   bytes32 rules_ipfs;
69   mapping(bytes32 => State)  elements;
70   mapping(bytes32 => bytes32) allValues;
71   struct Voter{bool voted; bool vote;}
72   struct Proposal{
73      bytes32 proposedHash;
74      uint quorum;
75      uint actualVotes;
76   }
77   function executeMessage(
78      string memory messageToExecute,
79      string[] memory inputs
80   ) public{...}
81
82   function setVariables(
83      bytes32[] memory names,
84      bytes32[] memory values,
85      string memory messageID
86   ) public{...}
87
88   function createProposal(
89      string memory _proposalHashRules,
90      string memory _proposalHashIds
91   ) public{...}
92
93   function voteProposal(
94      bool _vote
95   ) public{...}
96   ...
97  }
```

For interested readers, we provide the full code[6] of the factory and instance smart contracts.

5. FlexChain demonstration

In this section, we demonstrate the feasibility of the approach using the FlexChain tool to execute a healthcare-related case study. We provide a description of the X-rays exam process before and after the updated laws about Covid restrictions. Then, we show the instantiation, updating, and execution phases highlighting the main functionalities of the tool.

5.1. Case study

Fig. 10 shows the process that a patient must follow to schedule and do the X-rays examination. The patient, owning the related medical prescription, asks for an appointment with the radiology department that will check the availability of the ward. If there is a free date, the appointment is confirmed and the response is sent to the patient; otherwise, the availability is checked again. The patient can perform the physical check-in by providing the appointment identifier that is controlled by the radiology. If the appointment is confirmed, the X-rays exam is done and the results are provided to the ward that will compile a final report.

Fig. 11 reports instead the updated process dealing with the restrictions imposed by national law after the Covid pandemic. In particular, the main change regards the physical check-in that must follow additional steps. This requires verification of the temperature and of the vaccine certification. If the temperature is under the threshold and the certification is valid, the radiology will finally check the appointment identifier of the patient and return back the confirmation. The last phase is similar to the previous version of the model, where the exam report is analysed and provided to the patient.

5.2. The FlexChain tool

We report here the description of the FlexChain tool[7] by providing an example of the different phases of the approach.

Instantiation phase. The first operation in the proposed approach is the instantiation of a model. This functionality can be triggered directly from the FlexChain frontend (Fig. 13) after the design of the model using the provided modeller based on the Chor-js implementation [34]. The modeller, indeed, permits to create, upload, download and deploy a choreography model. Using the *deploy* button on the interface, the instantiation process starts as reported in Fig. 12. The first step consists of the automatic generation of the corresponding rules and their storage inside the IPFS. This generates the hash of the rules that is sent back to the frontend. At this point, the factory smart contract is invoked passing the generated IPFS hash and the *quorum* that must be reached in an eventual update phase. The factory contract will use a fixed template to generate a new instance contract containing all the information relative to the uploaded model, such as the list of the choreography elements. The address of the newly generated contract is then stored in the factory terminating the instantiation phase.

[6] https://bitbucket.org/proslabteam/flexchain_v2/src/master/Smart%20Contracts/.

[7] The tool is available at https://pros.unicam.it/flexchain/ while its code is accessible at https://bitbucket.org/proslabteam/flexchain_v2/.

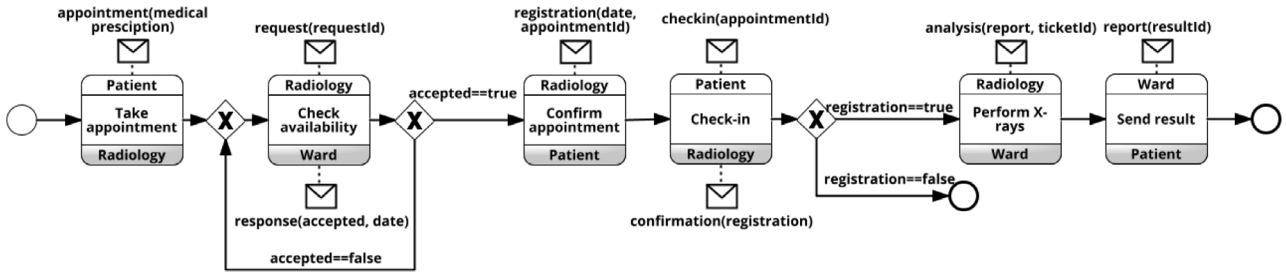


Fig. 10. X-rays exam process.

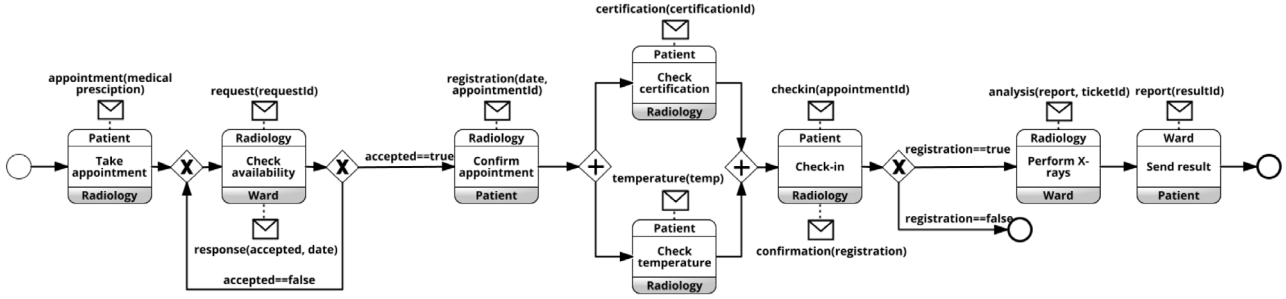


Fig. 11. X-rays exam process updated with Covid procedure.

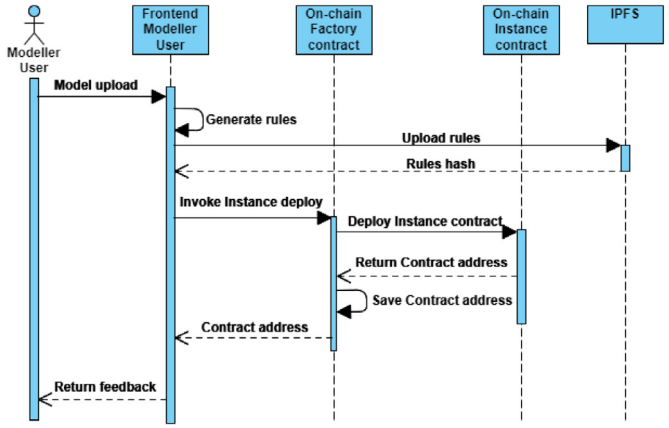


Fig. 12. Sequence diagram of the instantiation phase.

Update phase. In the update phase, it is possible to make runtime changes in order to react to unexpected situations or to optimise the running process. The FlexChain tool provides a dedicated page (Fig. 15) where the user can design the updates. The first step consists to upload the model corresponding to the original choreography to change; the model will be then visualised inside the two panels of the page. The upper one is a viewer and it only contains the imported choreography, the lower one instead is a modeller and it can be used by the proposer user to change the BPMN elements creating the updated version of the model. After this, it is possible to click the *View Changes* button to show the differences between the two models that will be coloured in green (added elements) or in red (removed elements). To effectively upload the changes in the blockchain, first, the user has to select the related smart contract using the drop-down list. This is possible thanks to the factory contract that during the instantiation associates the process names to their addresses (acting as a monitor) so as to be easily retrieved. After selecting the contract, a first control comparing the imported choreography elements (not the updated ones) with those stored inside the instance contract is made. In case they do not match, an error is shown to the user, otherwise, the *Update rules* button is enabled to activate the update process described in Fig. 14. The frontend generates the new rules for the updated elements and stores them inside the IPFS. After that, the voting procedure starts for approving the new rules and adopting them. Indeed, whenever a new update is made in the blockchain, it does not immediately change the state of the contract but it comes as an *update proposal* that must be approved by the other voter participants. When a proposal is created, an event is emitted and it is captured by the other participants that can vote using their frontend to approve, or not, the proposal. When the quorum is reached, the previously created IPFS hash is permanently written in the blockchain and the participants are informed.

Execution phase. In the execution phase (Fig. 16), the participants of a choreography instance can interact by sending messages according to the flow prescribed by the model. Fig. 17 shows the execution page, where the user can select the choreography to execute from the list of the available ones. At this point, the user can select one message to execute from the list of available ones reported in the drop-down menu (automatically retrieved from the blockchain). The required parameters can be inserted in the dedicated form provided by the FlexChain user interface. Finally, the button *Execute message* sends the request to the smart contract that emits an event containing the message to execute, the IPFS hash of the rules, and the inserted inputs. The off-chain processor captures this event, reads the rules from the IPFS, and executes the one related to the message. If the conditions are successfully evaluated, the inputs are written in the smart contract producing an instance state update. This returns also feedback to the user informing them about the status of the execution. To notice, in the proposed scenario we have implemented the off-chain processor as a Spring Boot server while the user interface is a React application. However, this is only one of the possible implementations since our approach fits also other technological choices.

Case study cost analysis. We report in Table 1 the cost analysis made on the execution of the proposed case study.[8] In particular,

[8] The entire execution is available at https://bit.ly/3wfKm8U.

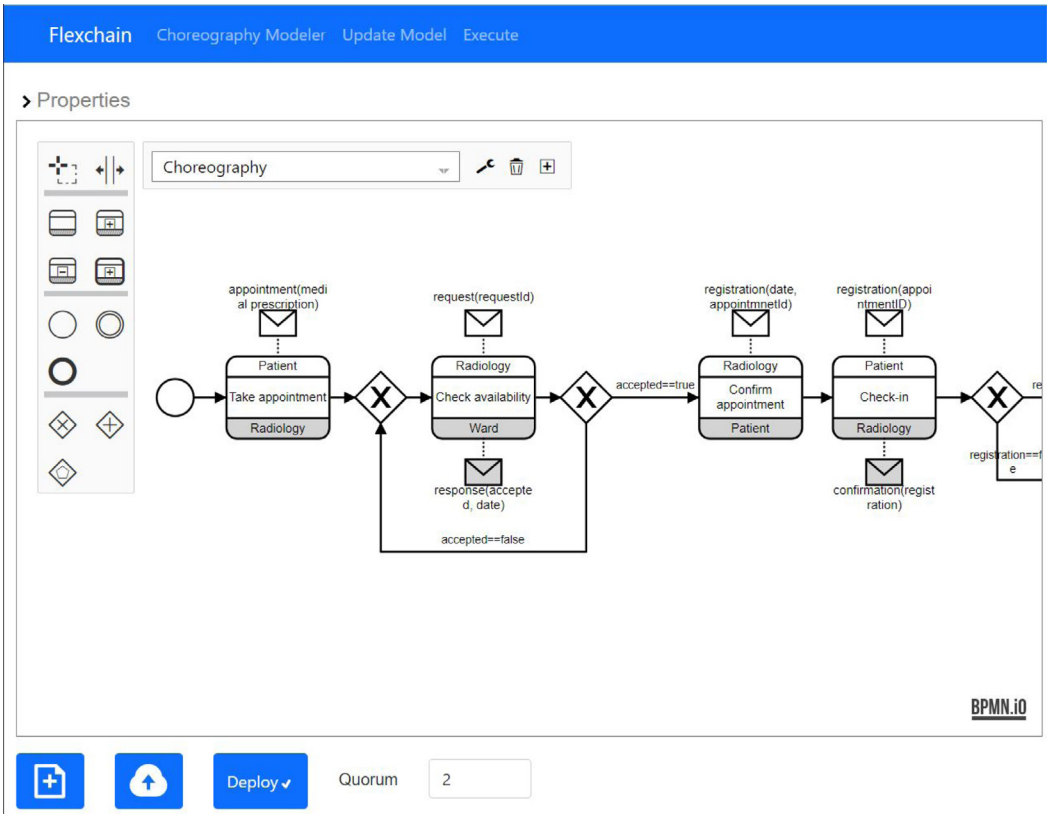


Fig. 13. Modelling page.

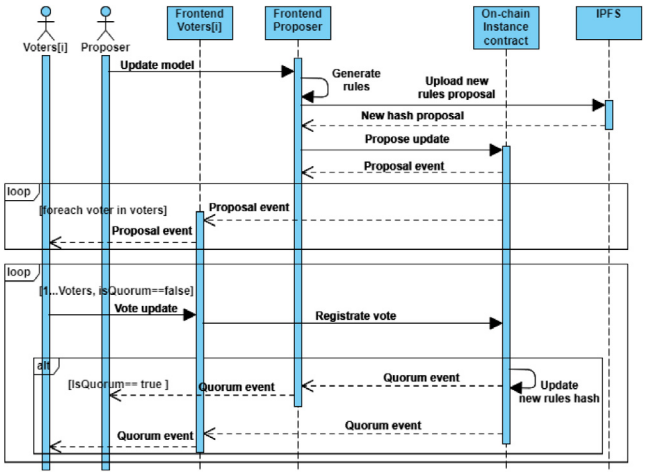


Fig. 14. Sequence flow of the updating phase.

we notice that the main costs are related to the deployment of the factory contract (around 1,8 million units of gas) and to the instantiation of the template for a new choreography (around 1,2 million units of gas). However, the factory is deployed only once since it is responsible for deploying choreography contracts. The choreography contract, instead, is deployed every time a new choreography is uploaded and it will store the identifier of the IPFS location of the rules. This step requires a transaction consuming a significantly lower amount of gas (around 158,000 units of gas). Once the choreography is deployed, it can be changed by means of an updated proposal to the contract (around 212,000 units of gas) that must be approved by the other participants

Table 1
Cost analysis for the X-rays process.

Transaction name	Gas used
Factory contract deploy	1,879,537
Template contract instantiation	1,229,125
IPFS hash upload	158,776
Update proposal	212,114
Vote for proposal	77,758
Average message execution	53,309

(around 77,000 units of gas). Compared to the deployment operations, the update one comes with lower gas consumption. Indeed, thanks to our approach, instead of deploying new contracts every time a change is done, it is only necessary to complete the voting mechanism storing the new IPFS referrals. Finally, users can participate in the execution phase by sending messages. This is the less consuming action since it requires only around 53,000 units of gas. Indeed, it is important to maintain low consumption due to the high frequency of exchanged messages regarding the deployment operations. In general, we consider the overall approach gas efficient, since only one-time operations require a major consumption of gas, in favour of more efficient execution.

6. Related works

The challenge of providing flexibility during the execution of process-aware information systems is largely discussed in the literature and many process management systems supporting flexibility are currently developed [15,35]. Also, different approaches are provided in many contexts; for example, in [36] the authors proposed a real-time system for dealing with the flexibility of cloud service workflows. In particular, they combine BPMN and UML (Unified Modeling Language) that are used

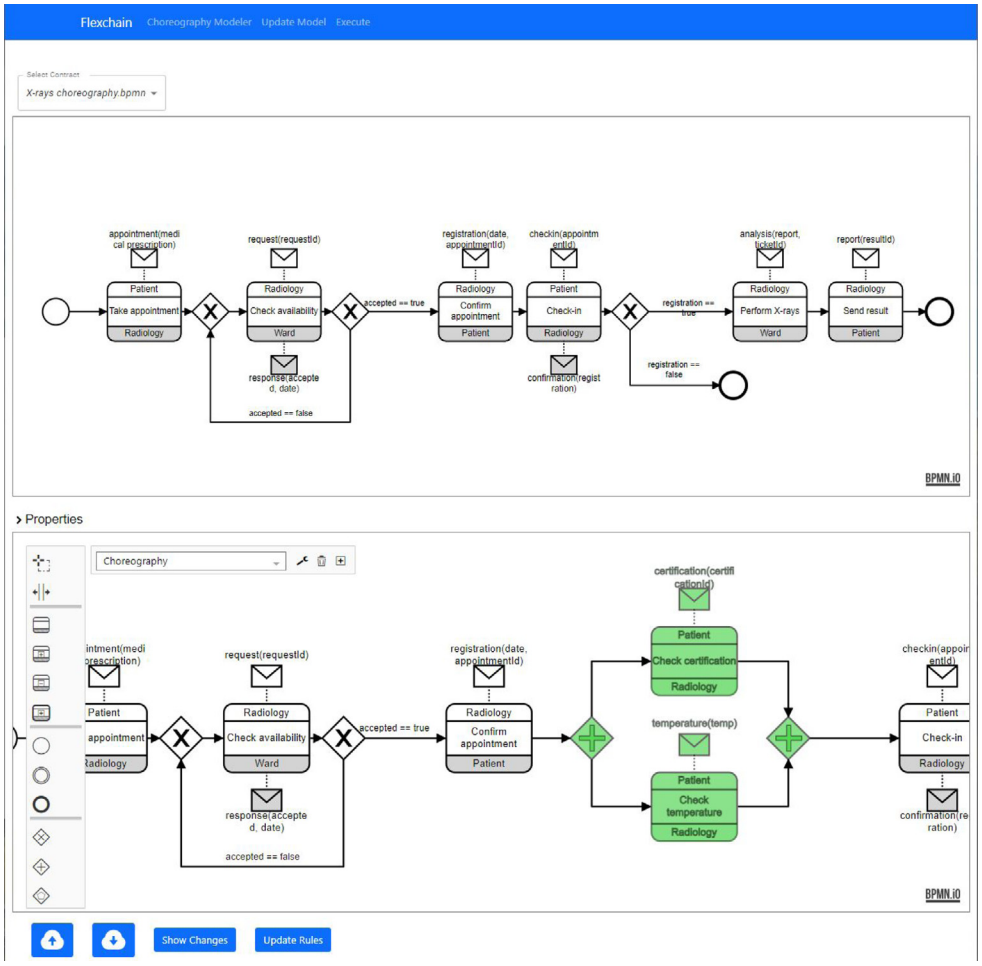


Fig. 15. Updating page.

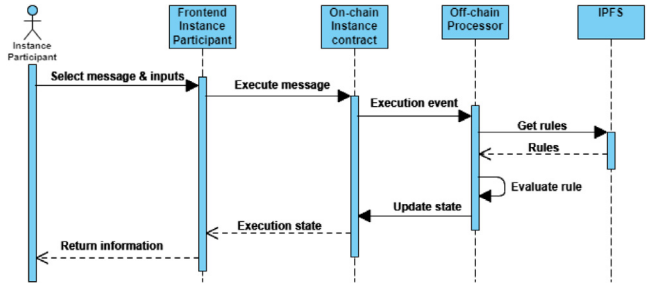


Fig. 16. Sequence flow of the execution phase.

to represent the functional and behavioural views of the cloud workflows. During the execution phase, the proposed system uses defined QoS (Quality of Service) attributes to detect and correct an anomaly whenever it occurs.

Differently, support for flexibility in business process modelling and execution can be achieved in different ways. In [37] the authors proposed an extension for BPMN, whose main objective is to obtain controlled flexibility. By exploiting this extension, process designers can model which business process element can change and under which conditions, by using specific expressions.

The work in [38], instead, focuses on versioning. It proposes an approach supporting dynamic changes in business processes, and discusses issues on version management for process changes. The proposed solution is based on change patterns that are concretely supported by a process designer extending BPEL (Business Process Execution Language).

Finally, similar to our work, in [39] dynamic business processes are executed using rules that are selected dynamically based on the surrounding context. In this way, it is possible to support changes at run-time by executing new actions and rules coming from the updated context.

All the approaches described above show how it is possible to obtain flexibility in different kinds of business processes. However, all of them significantly differ from our proposal, since our main focus concerns the execution of blockchain-based processes. In fact, we aim at guaranteeing trust while we achieve flexibility. This topic is still almost unexplored, only a few approaches are available.

In [40] the authors propose a dynamic role binder for run-time choice of sub-processes. They provide functionalities for the binding and unbinding of actors to a role in a dynamic way, supported by consistency verification and based on agreements. However, the proposal is mainly a run-time selection of already designed situations, with a focus on dynamic role-binding. Our approach, instead, wants to provide flexibility during the execution of the business process, without the need of specifying in the modelling phase the elements of the model that could be modified at execution time.

In [41], the work focuses on reaching flexibility on the technologies executing business processes. The resulting system is indeed developed with on-chain and off-chain components based on federated blockchains. However, here the focus is on the

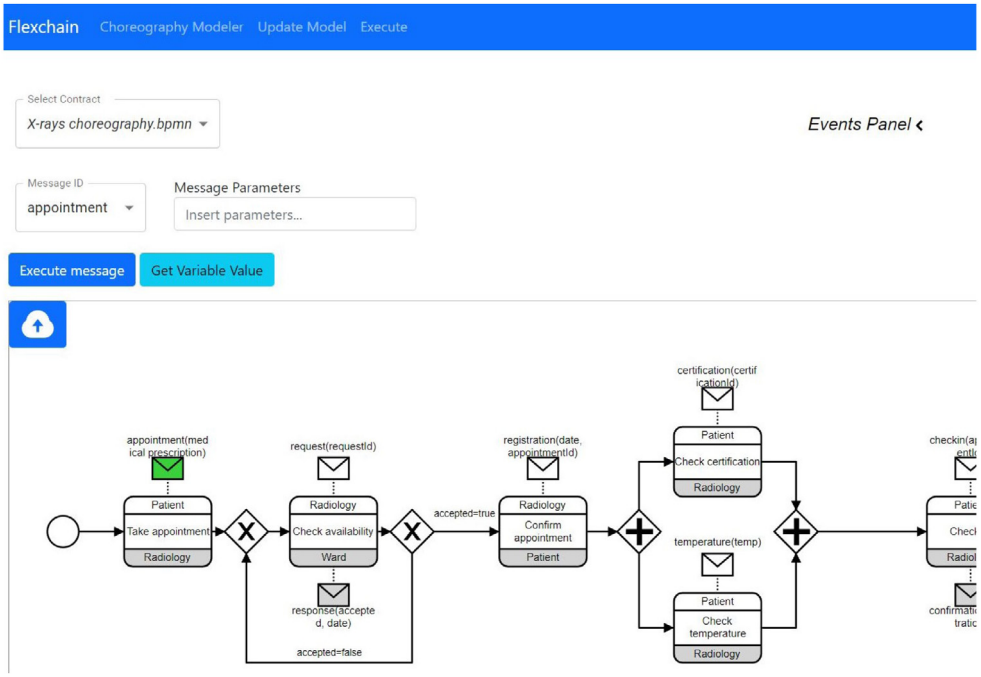


Fig. 17. Execution page.

Table 2
Table of identified related works and their characteristics.

Source	BPMN	Blockchain	Flexibility
[36]	✓	✗	Scaling deployed services during the workflow
[37]	✓	✗	Modelling and executing controlled flexibility
[38]	✗	✗	Dynamic process version selection
[39]	✓	✗	Rule- and context-based modelling and simulation
[40]	✓	✓	Run-time selection of predesigned elements
[41]	✓	✓	Dynamic blockchain selection
[42]	✓	✓	Upgradeability and versioning approach

flexible interactions between different technologies. In our case, instead, we want to achieve flexibility in the execution of the business process itself and not in the underlying technologies.

Finally, in [42] the issue of upgradeability is faced. The authors investigate how, starting from a collaboration diagram, it is possible to upgrade the smart contracts representing the business process, its state and its participants. They use a combination of static and dynamic contracts for the storage of logic and states bringing high costs and a high level of complexity. Also in this case, the main difference with our work consists in the motivation. Indeed, we aim at providing a flexible execution at run-time of active instances of business processes, while these authors focused on the dynamic versioning of inactive instances.

In Table 2, we summarise the analysed works concerning flexibility and, in particular, we consider three main characteristics: (i) if the work explicitly uses BPMN diagrams for modelling, (ii) if the work relies on blockchain, and (iii) which flexibility aspects the work supports, and how it implements them. The table shows that in most cases flexibility is achieved in a setting that does not rely on blockchain technology, hence without facing all issues raised by the immutability of the blockchain. In the approaches that, instead, adopt blockchain for a trusted environment, the concept of flexibility they consider differs from work to work. In [41], the authors consider flexibility as the possibility to select different blockchains, while in [40] the authors provide the possibility to bind actors and control-flow decisions at run-time. In [42], the work focuses on the smart contract upgrading mechanism, thus proposing a versioning approach. Differently, in our work we focused on a richer flexibility mechanism for business processes, whose business logic and execution state is stored in the blockchain, supporting a voting mechanism that permits arbitrary changes in the model at run-time.

Contributions. This work extends and improves the FlexChain approach and tool proposed in [18]. We have significantly revised the methodological phases, by adding some core components, such as IPFS, and introducing a factory contract to replace the old one. In particular, the introduction of the IPFS as external storage reduces costs, thus optimising the overall approach. The factory smart contract, instead, allows generating new instance contracts, facilitating their management and acting also as a monitor. Furthermore, from a methodological point of view, we have introduced a quorum mechanism, supported by the smart contracts and by the FlexChain tool. The quorum allows preventing the creation of unauthorised and malicious changes by requiring approval from the participants of the process. All these novelties were also supported by the new FlexChain tool, which has been re-implemented as a web application available online. Inside it, we have integrated a modeller for choreographies and several panels containing functionalities for creating, interacting and modifying a choreography instance. Finally, we have extended the discussion of related works, by adding new comparisons and enriching the already existing ones.

7. Conclusion

In this work, we propose an approach for the flexible execution of multi-party business processes based on blockchain. In these last years, more and more works exploited blockchain technology for the development of multi-party business processes. Indeed, thanks to its characteristics of disintermediation, transparency and distribution, blockchain is seen as disruptive in many different scenarios. However, while traditional (non-blockchain-based) business processes are supported by a large variety of frameworks that implement their entire life cycle, the introduction of blockchain brings many new challenges. In particular, this

works focuses on the flexible aspect of the business process execution. Indeed, it could happen that due to unexpected situations (e.g., the pandemic or the involvement of a new stakeholder) the process has to be changed at run-time. While many solutions are available in traditional settings, flexibility remains almost unexplored in blockchain-based ones. The main reason comes from the immutability of blockchain and its programs (i.e., smart contracts) which brings additional complexity since at run-time it is not possible to modify already-defined business rules. For this reason, our approach aims at providing a framework for dealing with this challenge while maintaining the fundamental properties of trust, transparency and decentralisation. To deal with the immutability of the blockchain, the proposed solution decouples the state of the process (stored on-chain) from the execution logic (stored off-chain). Indeed, while the first is a set of data stored in a smart contract, the second one is expressed as rules and they are stored in the IPFS. This allows to change them at run-time without having to modify the on-chain code that, in other cases, results impossible.

To demonstrate its feasibility, we applied the approach relying on the Ethereum blockchain. We also implemented a tool supporting the entire life-cycle of BPMN choreographies, from their modelling to their execution and update, and we experimented on a healthcare case study.

It is worth noticing that our work does not handle time constraints as modelling primitives (e.g., timer events) in choreography diagrams (see Fig. 1). As a result, FlexChain does not support the development of real-time applications. Their implementation in FlexChain is also limited by the latency and throughput of the underlined blockchain technology. In particular, the performance of the Ethereum public blockchain used in FlexChain is affected by many factors, such as the consensus algorithm, the network congestion, the size of the payload for each transaction, and the correlated fees. However, FlexChain is not limited to the Ethereum technology but is open to every EVM-based implementation. For this reason, the performance of the overall approach could be improved by changing the underlined technology and moving, for example, towards a private Ethereum network or a more scalable platform using a Layer 2 solution. We plan as future work to integrate a Layer 2 technology (e.g., Polygon) in FlexChain to improve our approach's overall performance. We also intend to investigate possible solutions for supporting the development of real-time applications in FlexChain.

As future work, we also plan to introduce mechanisms able to evaluate the changes proposed by a user in the model in order to prevent the application of inconsistent updates that invalidate the execution. Finally, we intend to define a real-time monitor that could help the users to reason about the current state of the execution, thus providing a more valuable experience.

CRediT authorship contribution statement

Flavio Corradini: Conceptualization, Resources, Supervision, Funding acquisition. **Alessandro Marcelletti:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Visualization. **Andrea Morichetta:** Conceptualization, Methodology, Software, Validation, Investigation, Writing – original draft, Visualization. **Andrea Polini:** Conceptualization, Methodology, Resources, Writing – review & editing, Supervision, Project administration. **Barbara Re:** Conceptualization, Methodology, Resources, Writing – review & editing, Supervision, Project administration. **Francesco Tiezzi:** Conceptualization, Methodology, Resources, Writing – review & editing, Supervision, Project administration.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

Data will be made available on request.

Acknowledgment

This work was partially supported by the project SERICS (PE00000014) under the NRRP MUR program funded by the EU - NextGenerationEU.

References

[1] OMG, Business process model and notation (BPMN), 2011, URL https://www.omg.org/spec/BPMN/2.0/PDF/.

[2] J. Mendling, I. Weber, W.M.P. van der Aalst, J. vom Brocke, C. Cabanillas, F. Daniel, S. ren Debois, C.D. Ciccio, M. Dumas, S. Dustdar, A. Gal, L. García-Bañuelos, G. Governatori, R. Hull, M.L. Rosa, H. Leopold, F. Leymann, J. Recker, M. Reichert, H.A. Reijers, S. Rinderle-Ma, A. Solti, M. Rosemann, S. Schulte, M.P. Singh, T. Slaats, M. Staples, B. Weber, M. Weidlich, M. Weske, X. Xu, L. Zhu, Blockchains for business process management - challenges and opportunities, ACM Trans. Manag. Inf. Syst. 9 (1) (2018) 4:1–4:16, http://dx.doi.org/10.1145/3183367.

[3] S. Porru, A. Pinna, M. Marchesi, R. Tonelli, Blockchain-oriented software engineering: challenges and new directions, in: Proceedings of the 39th International Conference on Software Engineering, IEEE Computer Society, 2017, pp. 169–171, http://dx.doi.org/10.1109/ICSE-C.2017.142.

[4] S. Curty, F. Härer, H. Fill, Blockchain application development using model-driven engineering and low-code platforms: A survey, in: Enterprise, Business-Process and Information Systems Modeling, in: Lecture Notes in Business Information Processing, vol. 450, Springer, 2022, pp. 205–220, http://dx.doi.org/10.1007/978-3-031-07475-2_14.

[5] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi, Engineering trustable and auditable choreography-based systems using blockchain, ACM Trans. Manag. Inf. Syst. 13 (3) (2022) 31:1–31:53, http://dx.doi.org/10.1145/3505225.

[6] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi, Engineering trustable choreography-based systems using blockchain, in: 35th ACM/SIGAPP Symposium on Applied Computing, ACM, 2020, pp. 1470–1479, http://dx.doi.org/10.1145/3341105.3373988.

[7] O. López-Pintado, L. García-Bañuelos, M. Dumas, I. Weber, A. Ponomarev, Caterpillar: A business process execution engine on the ethereum blockchain, Softw. Pract. Exp. 49 (7) (2019) 1162–1193, http://dx.doi.org/10.1002/spe.2702.

[8] A.B. Tran, Q. Lu, I. Weber, Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management, in: Proceedings of the Dissertation Award, Demonstration, and Industrial Track At BPM, in: CEUR Workshop Proceedings, vol. 2196, CEUR-WS.org, 2018, pp. 56–60.

[9] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, E. Scala, F. Tiezzi, Model-driven engineering for multi-party business processes on multiple blockchains, Blockchain: Res. Appl. 2 (3) (2021) 100018, http://dx.doi.org/10.1016/j.bcra.2021.100018.

[10] F. Stiehle, I. Weber, Blockchain for business process enactment: A taxonomy and systematic literature review, in: Business Process Management: Blockchain, Robotic Process Automation, and Central and Eastern Europe Forum - BPM 2022 Blockchain, RPA, and CEE Forum, in: Lecture Notes in Business Information Processing, vol. 459, Springer, 2022, pp. 5–20, http://dx.doi.org/10.1007/978-3-031-16168-1_1.

[11] R. Cognini, F. Corradini, S. Gnesi, A. Polini, B. Re, Research challenges in business process adaptability, in: Symposium on Applied Computing, ACM, 2014, pp. 1049–1054, http://dx.doi.org/10.1145/2554850.2555055.

[12] W.M. Van der Aalst, Business process management: a comprehensive survey, Int. Sch. Res. Not. (2013) 1–37.

[13] R. Cognini, F. Corradini, S. Gnesi, A. Polini, B. Re, Business process flexibility-a systematic literature review with a software systems perspective, Inf. Syst. Front. 20 (2) (2018) 343–371, http://dx.doi.org/10.1007/s10796-016-9678-2.

[14] M. Reichert, B. Weber, Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies, Springer, 2012.

[15] A. Mejri, S.A. Ghanouchi, R. Martinho, Evaluation of process modeling paradigms enabling flexibility, Procedia Comput. Sci. 64 (2015) 1043–1050, http://dx.doi.org/10.1016/j.procs.2015.08.514.
[16] Q. Zheng, Y. Li, P. Chen, X. Dong, An innovative IPFS-based storage model for blockchain, in: 2018 IEEE/WIC/ACM International Conference on Web Intelligence, WI, 2018, pp. 704–708, http://dx.doi.org/10.1109/WI.2018.000-8.
[17] Y. Chen, H. Li, K. Li, J. Zhang, An improved P2P file system scheme based on IPFS and blockchain, in: 2017 IEEE International Conference on Big Data (Big Data), 2017, pp. 2652–2657, http://dx.doi.org/10.1109/BigData.2017.8258226.
[18] F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi, Flexible execution of multi-party business processes on blockchain, in: International Workshop on Emerging Trends in Software Engineering for Blockchain, IEEE, 2022, pp. 25–32, http://dx.doi.org/10.1145/3528226.3528369.
[19] E. Verbeek, D. Fahland, CPN IDE: An extensible replacement for CPN tools that uses access/CPN, in: ICPM Doctoral Consortium and Demo Track, in: CEUR Workshop Proceedings, vol. 3098, CEUR-WS.org, 2021, pp. 29–30.
[20] Y. Thierry-Mieg, Symbolic model-checking using ITS-tools, in: TACAS, in: Lecture Notes in Computer Science, vol. 9035, Springer, 2015, pp. 231–237.
[21] M.S. Arbabi, C. Lal, N.R. Veeraragavan, D. Marijan, J.F. Nygard, R. Vitenberg, A survey on blockchain for healthcare: Challenges, benefits, and future directions, IEEE Commun. Surv. Tutor. (2022) 1, http://dx.doi.org/10.1109/COMST.2022.3224644.
[22] S. Rani, H. Babbar, G. Srivastava, T.R. Gadekallu, G. Dhiman, Security framework for internet of things based software defined networks using blockchain, IEEE Internet Things J. (2022) 1, http://dx.doi.org/10.1109/JIOT.2022.3223576.
[23] S. Hakak, W.Z. Khan, G.A. Gilkar, B. Assiri, M. Alazab, S. Bhattacharya, G.T. Reddy, Recent advances in blockchain technology: A survey on applications and challenges, Int. J. Ad Hoc Ubiquitous Comput. 38 (1–3) (2021) 82–100.
[24] A. Abraham, Rule-based expert systems, Handb. Meas. Syst. Des. (2005).
[25] B.G. Buchanan, R.O. Duda, Principles of rule-based expert systems, in: Adv. Comput., Vol. 22, Elsevier, 1983, pp. 163–216, http://dx.doi.org/10.1016/S0065-2458(08)60129-1.
[26] N. Kumar, D.D. Patil, V.M. Wadhai, Rule based programming with drools, Int. J. Comput. Sci. Inf. Technol. 2 (3) (2011) 1121–1126.
[27] J. Benet, IPFS - content addressed, versioned, P2P file system, 2014, CoRR abs/1407.3561.
[28] X. Xu, C. Pautasso, L. Zhu, Q. Lu, I. Weber, A pattern collection for blockchain-based applications, in: Proceedings of the 23rd European Conference on Pattern Languages of Programs, EuroPLoP '18, ACM, 2018, pp. 3:1–3:20, http://dx.doi.org/10.1145/3282308.3282312.
[29] D. Khan, L.T. Jung, M.A. Hashmani, Systematic literature review of challenges in blockchain scalability, Appl. Sci. 11 (20) (2021) http://dx.doi.org/10.3390/app11209372, URL https://www.mdpi.com/2076-3417/11/20/9372.
[30] R. Mühlberger, S. Bachhofner, E.C. Ferrer, C.D. Ciccio, I. Weber, M. Wöhrer, U. Zdun, Foundational oracle patterns: Connecting blockchain to the off-chain world, in: Business Process Management: Blockchain and Robotic Process Automation Forum - BPM 2020 Blockchain and RPA Forum, Seville, Spain, September 13-18, 2020, Proceedings, in: Lecture Notes in Business Information Processing, vol. 393, Springer, 2020, pp. 35–51, http://dx.doi.org/10.1007/978-3-030-58779-6_3, URL https://doi.org/10.1007/978-3-030-58779-6_3.
[31] X. Li, P. Jiang, T. Chen, X. Luo, Q. Wen, A survey on the security of blockchain systems, Future Gener. Comput. Syst. 107 (2020) 841–853, http://dx.doi.org/10.1016/j.future.2017.08.020.
[32] A. Bertolino, E. Marchetti, A. Morichetta, Adequate monitoring of service compositions, in: B. Meyer, L. Baresi, M. Mezini (Eds.), Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering, ESEC/FSE'13, Saint Petersburg, Russian Federation, August 18-26, 2013, ACM, 2013, pp. 59–69.
[33] I. Compagnucci, F. Corradini, F. Fornari, B. Re, Trends on the usage of BPMN 2.0 from publicly available repositories, in: Perspectives in Business Informatics Research, Springer International Publishing, 2021, pp. 84–99.
[34] J. Ladleif, A. von Weltzien, M. Weske, chor-js: A modeling framework for BPMN 2.0 choreography diagrams, in: Proceedings of the ER Forum and Poster & Demos Session, in: CEUR Workshop Proceedings, vol. 2469, CEUR-WS.org, 2019, pp. 113–117.
[35] H. Schonenberg, R. Mans, N. Russell, N. Mulyar, W.M.P. van der Aalst, Process flexibility: A survey of contemporary approaches, in: Advances in Enterprise Engineering I, in: Lecture Notes in Business Information Processing, vol. 10, Springer, 2008, pp. 16–30, http://dx.doi.org/10.1007/978-3-540-68644-6_2.
[36] I.B. Fraj, Y.B. Hlaoui, L.B. Ayed, A control system for managing the flexibility in BPMN models of cloud service workflows, in: International Conference on Cloud Computing, IEEE, 2020, pp. 537–543, http://dx.doi.org/10.1109/CLOUD49709.2020.00081.
[37] R. Martinho, D. Domingos, J. Varajão, CF4BPMN: a BPMN extension for controlled flexibility in business processes, Procedia Comput. Sci. 64 (2015) 1232–1239, http://dx.doi.org/10.1016/j.procs.2015.08.509.
[38] D. Kim, M. Kim, H. Kim, Dynamic business process management based on process change patterns, in: International Conference on Convergence Information Technology, IEEE, 2007, pp. 1154–1161, http://dx.doi.org/10.1109/ICCIT.2007.91.
[39] O. Vasilecas, D. Kalibatiene, D. Lavbič, Rule-and context-based dynamic business process modelling and simulation, J. Syst. Softw. 122 (2016) 1–15, http://dx.doi.org/10.1016/j.jss.2016.08.048.
[40] O. López-Pintado, M. Dumas, L. García-Bañuelos, I. Weber, Controlled flexibility in blockchain-based collaborative business processes, Inf. Syst. 104 (2022) 101622, http://dx.doi.org/10.1016/j.is.2020.101622.
[41] M. Adams, S. Suriadi, A. Kumar, A.H. ter Hofstede, Flexible integration of blockchain with business process automation: A federated architecture, in: International Conference on Advanced Information Systems Engineering, in: LNBIP, vol. 386, Springer, 2020, pp. 1–13, http://dx.doi.org/10.1007/978-3-030-58135-0_1.
[42] P. Klinger, L. Nguyen, F. Bodendorf, Upgradeability concept for collaborative blockchain-based business process execution framework, in: Conference on Blockchain, in: LNCS, vol. 12404, Springer, 2020, pp. 127–141, http://dx.doi.org/10.1007/978-3-030-59638-5_9.

Flavio Corradini is a Full Professor of Computer Science at the University of Camerino. He received a Laurea Degree in Computer Science from the University of Pisa and a Ph.D. in Computer Engineering from the University of Rome "La Sapienza". He has been the Director of the Mathematics and Computer Science Department, President of the digital services and information systems center of the University of Camerino, Coordinator of the Computer Science Studies of the University of Camerino, and rector of the University of Camerino. His main research activities are in the area of formal specification, verification of concurrent, distributed, and real-time systems.

Alessandro Marcelletti is Ph.D. Student in Computer Science and Mathematics at the University of Camerino. He received his Master's Degree from the University of Camerino in 2019. His research interests refer to the area of Blockchain technology and its integration into the business process management discipline. He also explores the benefits of Blockchain for IoT, creating new forms of trusted systems. His main results involve creating tools and development methodologies based on Blockchain. He also acts as Co-Organiser and speaker of the International Winter School on Blockchain Technology and Applications: Hyperledger.

Andrea Morichetta is an Assistant Professor of Computer Science at the University of Camerino. He received his Ph.D. in Computer Decision and Systems Science from IMT School for Advanced Studies Lucca in 2016. His research interests are mainly in the area of formal methods and software engineering applied to distributed systems, business process management, and blockchain technologies. In the last years, Morichetta has focused his interest on distributed ledger technologies and blockchain application development, with a particular focus on model-driven approaches.

Andrea Polini is an Associate Professor at the University of Camerino (UNICAM). His research interests are in the area of Software Engineering, in particular on Modelling methods and Quality Assurance strategies for Complex Software Systems. He is a coordinating member of the PROS Lab at UNICAM. He was a researcher at ISTI-CNR in Pisa and got a Ph.D. in Computer Engineering from Scuola Superiore Sant'Anna in Pisa. His research has always been linked to his participation in many EU research projects, and he has been Project Scientific Leader for the EU Collaborative Project Learn PAd.

Barbara Re is an Associate Professor of Computer Science at the University of Camerino. She received her Ph.D. in Information Science and Complex Systems from the University of Camerino. Her research interests refer to the area of Business Process Management from modelling to analysis. Particular attention is paid to the use of formal methods as methodological and automatic tools for developing high-quality process-aware information systems. She was involved in multidisciplinary research projects collaborating with national and international research institutes and companies.

Francesco Tiezzi is an Associate Professor of Computer Science at the University of Florence. He received the Laurea degree from the University of Florence and the Ph.D. degree from the same university. His research activities focus on methodologies and tools, possibly based on formal methods, for modelling, verifying, programming, and deploying distributed systems. Particular attention is paid to the definition of formal bases for solutions supporting service-oriented, cloud, and autonomic computing and, more recently, business process management and blockchain.