Journal Pre-proof

Model-Driven Engineering for Multi-Party Business Processes on Multiple Blockchains

Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, Emanuele Scala, Francesco Tiezzi francesco

PII: S2096-7209(21)00013-0

DOI: https://doi.org/10.1016/j.bcra.2021.100018

Reference: BCRA 100018

To appear in: *Blockchain: Research and Applications*

Received Date: 31 December 2020

Revised Date: 10 May 2021

Accepted Date: 7 June 2021

Please cite this article as: F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, E. Scala, F. Tiezzi francesco, Model-Driven Engineering for Multi-Party Business Processes on Multiple Blockchains, *Blockchain: Research and Applications*, https://doi.org/10.1016/j.bcra.2021.100018.

Model-Driven Engineering for Multi-Party Business Processes on Multiple Blockchains

Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta*, Andrea Polini, Barbara Re, Emanuele Scala, Francesco Tiezzi

{flavio.corradini, alessand.marcelletti, andrea morichetta, andrea.polini, barbara.re, emanuele.scala, francesco tiezzi}@unicam.it

*Corresponding author

Model-Driven Engineering for Multi-Party Business Processes on Multiple Blockchains

Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta*, Andrea Polini, Barbara Re, Emanuele Scala, Francesco Tiezzi

{flavio.corradini, alessand.marcelletti, andrea morichetta, andrea.polini, barbara.re, emanuele.scala, francesco tiezzi}@unicam.it

Abstract

As a disruptive technology, the blockchain is continuously finding novel application contexts, bringing new opportunities and radical changes. In this paper, we use blockchain as a communication infrastructure to support multi-party business processes. In particular, through smart contracts specifically generated by the mentioned business process, it is possible to derive a trustable infrastructure enabling the interaction among parties. Moreover, the emergence of different blockchain technologies, satisfying different characteristics, gives the possibility to support the same business process dealing with different non-functional needs.

In this paper, we propose a novel engineering methodology supported by a practical framework called Multi-Chain. It permits to derive, using a model-driven strategy, a blockchain-based infrastructure, that can be deployed over a specific blockchain technology (e.g. Ethereum or Hyperledger Fabric). The objective is to permit the single definition and multiple deployments of the business process, to deliver the same functionalities, but satisfying different non-functional needs. In such a way, organisations willing to cooperate can select the multi-party business process and the blockchain technology they would like to use to satisfy their needs. Using Multi-Chain, they will be able to automatically derive from a BPMN choreography diagram a blockchain infrastructure ready to be used. This overcomes the need to get acquainted with many details of the specific technology.

Keywords: Blockchain, Model-Driven, Multi-Party, Choreography

*Corresponding author

1. Introduction

Blockchain technologies have been recently recognised as an effective means for the decentralised execution of multi-party business processes in the Business Process Management discipline [18]. The main characteristic that favoured their adoption refers to the possibility of guaranteeing the integrity and the immutability of exchanged messages, without relying on a central authority [5, 17]. Roughly, a multi-party business process establishes the rules that different organisations should follow to enable their interaction/integration, making possible that their informative systems can collaborate to reach a shared goal. So far, implementing such business processes considers two possible scenarios: introducing a central authority, usually called orchestrator, or coordinating the involved parties in a distributed manner. In a business context where transactions generally have an economic relevance, the parties not necessarily trust each other and then the centralised approach could be unsatisfactory. Indeed the orchestrator constitutes a centralisation point that could take actions to favour one of the parties. On the other hand, the purely distributed approach does not permit each party to fully observe the interactions performed by the other participants to check if they abide by the specified rules. It results that blockchain adoption enables the development of new forms of multi-party business processes. All participants, without relying on a central authority, can have a clear view of the ongoing system execution and can have tangible proofs of the actions performed by all participants.

Moreover, focusing on blockchain technologies, it is possible to distinguish between two broad categories. The first one refers to permissionless blockchains, like Ethereum [8], that consists of a public network without any restriction regarding access to the recorded transactions or the identity of the participants that can join the blockchain. These categories of blockchain suits in completely trustless environments, where privacy and sensible data [10] are not so many relevant aspects to hinder the adoption of this technology. The second category should be adopted when access to the stored data and the access of participants are restricted. Such kinds of blockchain are referred to as permissioned blockchains, like Hyperledger Fabric [2]. The two different technologies lead to different scenarios about the engineering of multi-party business processes we targeted. Indeed, there are situations

where process execution best suits in a permissionless scenario, for instance, with participants that can dynamically join. On the other hand, there are also scenarios where a multi-party business process execution best fits with a permissioned scenario with pre-defined participants and restricted access to data.

Being aware of Business Process Modelling Notation (BPMN) [19] emerging as a modelling language to describe and also to support the engineering multi-party business processes [1, 20], in this paper, we focus on the BPMN choreography diagram. It permits to describe the messages that have to be exchanged among the involved participants from a global perspective, without exposing any internal behaviour of the participants.

By relying on BPMN, we provide a model-driven methodology that permits to derive a run-time blockchain-based infrastructure enabling the execution of a multi-party business process. Specifically, starting from a BPMN choreography specification, we support the generation and deployment of proper infrastructure, based on smart contracts, that will enable the execution of the multi-party business process on Ethereum or Hyperledger Fabric, according to different non-functional characteristics as detailed in the next section. The proposed methodology is supported by a practical framework, named Multi-Chain, that allows the adoption of blockchain technologies easier.

Summing up, the contribution of the paper is twofold.

- A model-driven methodology for the generation of different blockchain infrastructures from the same model of a multi-party business process, that is a choreography diagram specification.

- The implementation of the methodology in a practical framework to enable the deployment on both Ethereum and Hyperledger Fabric.

The work presented here takes advantage of the results reported in the conference paper [7], where the ChorChain framework is introduced. In particular, in [7] we illustrate how smart contracts can be used to enable the adoption of the Ethereum blockchain to support a multi-party business process. The ChorChain framework, however, has been designed to fit only a single blockchain implementation, i.e. Ethereum, assuming specific needs. Here, we have revised, generalised and extended that approach in order to fit different requirements and multiple blockchain technologies. This results in a more abstract engineering methodology, supported by a practical framework

called Multi-Chain. Starting from the same high-level specification, it permits generating the low-level code specific for different blockchain platforms (at the time being, Ethereum and Hyperledger Fabric). This characteristic of the extended approach makes it reusable in a broader range of application scenarios.

The rest of the paper is organised as follows. Section 2 clarifies the motivations behind our work and introduces the running example we used. Section 3 includes a general presentation of the BPMN notation. Successively, Section 4 introduces the proposed methodology, Section 5 provides an overview of the smart contract generation, and Section 6 illustrates the Multi-Chain practical framework focusing on specific implementation details. Section 7 discusses performances, limitations and open challenges. Finally, Section 8 reviews relevant related works, and Section 9 concludes the paper by touching upon directions for future work.

2. Motivations

Blockchain technologies. In a distributed setting, usually, the parties can not have any guarantee about the performed interactions due to the untrusted nature of the participants. Blockchain technologies seemed to provide an effective solution. Through the specification of suitable smart contracts, the distributed parties could interact in a safe and trusted way through the blockchain. Indeed, all the interactions will be immutably stored and made available for successive scrutiny. In such a way, the trust problem, inherent within a multi-party distributed system, is clearly mitigated. The blockchain can therefore open interesting scenarios in the context of multi-party business processes execution. Indeed, this new technology allows overcoming the limits of current solutions for business processes execution (such as, e.g., the workflow engines Camunda[1] and Flowable[2]) given by the use of a third-party authority to interact with unknown parties. Also, they do not allow to realise a real distributed implementation, but rely on a centralised approach that may result in potential failure issues. This aspect is particularly relevant in an inter-organisational scenario.

Nonetheless, the adoption of blockchain introduces a further degree of complexity for those not familiar with such a technology. The current trend

[1] `https://camunda.com/`
[2] `https://flowable.com/`

towards introducing different blockchain technologies, with different characteristics, makes the situations even worse, with a proliferation of technologies to be acquired and learned.

For our purpose, it is possible to distinguish among the following characterisations.

- **Permissionless vs Permissioned**: a permissionless blockchain is an open network where participants can join, and leave the network without the need of any authorisation. A permissioned blockchain runs a ledger among a set of previously identified and authorised peers.
- **Auditability vs Confidentiality**: an auditable blockchain has an immutable and transparent nature, and it natively allows independent auditing over the stored data. On the contrary, a permissioned blockchain introduces confidentiality so that stored data are not visible to anyone. Moreover, it restricts the distribution of information only to authorised nodes.
- **High decentralisation vs Performance and scalability**: the usage of strong consensus algorithms allows to trust nodes previously unknown or not trusted, in a decentralised context. On the contrary, the introduction of access control mechanisms leads to a trusted network with higher scalability and transaction throughput
- **Anonymity vs Identity**: a blockchain technology could permit anyone to join the network without putting in place any access control mechanism. Trust over the stored data will be in any case guaranteed by the consensus algorithm. On the other hand, access to a blockchain can be restricted to authorised users introducing specific mechanisms. Consequently, it will be possible to associate identities with participants, and cryptographic credentials can be issued to new members. All communications can also be made use of authentication mechanisms.

In particular, in this paper, we consider two main blockchain technologies: Ethereum and Hyperledger Fabric. The two technologies present rather orthogonal characteristics and have been conceived for different application domains. Table 1 compares Ethereum and Hyperledger Fabric with respect to the list of properties presented above, as confirmed in [21, 26, 23, 29, 27].

Ethereum implementation	**Hyperledger Fabric implementation**
Permissionless	Permissioned
Auditability	Confidentiality
High Decentralisation	Performance and Scalability
Anonymity	Identity

Table 1: Blockchain characterisations.

The permissionless characteristic of blockchain, like Ethereum, guarantees a trusted and verifiable communication between untrusted and unknown organisations. At the same time, Ethereum lacks privacy, performance, and access controls. Permissioned blockchains, like Hyperledger Fabric, cover these aspects, leaving more freedom to the users in the network's organisation. In particular, this suits well when partial trust relationships between parties can be assumed.

In most cases, the right selection of the underlining blockchain technology for a given choreography scenario does not depend only on the system's behaviour. It is also influenced by the context in which the system will have to operate. This means that the same choreography model could be deployed in different situations within different blockchains technologies, depending on the level of trust required by the considered scenarios. The model-driven approach we propose brings clear benefits in such a context, permitting to alleviate the developer from deriving a codebase for each different technology.

Running example. To clarify our paper's objectives, we consider a simple scenario consisting of a multi-party business process that allows a customer to buy goods from a retailer. In the proposed example we refer to warehousing management, and in particular, the considered policy aims at reducing the warehousing costs. The retailer does not keep in the warehouse a high volume of goods and generally starts the acquisition process as soon as it receives a specific request. It is also possible that the customer's request indicates a specific producer to involve in the provisioning.

This example highlights an interaction scenario in which the requirements of trust and privacy change according to the contexts in which the system operates. Indeed, in a situation in which parties can trade freely without limits, the goods that are sold are easily reproducible, and prices are changed according to a standard agreement, there would not be issues related to the sharing of the information. In this scenario, at the same time, the traceabil-

ity of products could be a very important requirement for the customers to continuously check that the standard agreement is satisfied. In other contexts, the parties could operate in a close environment where, for business purposes, the participants are more interested in keeping private most of the information related to the products. In the following, we describe these two business contexts, to make clearer the different properties needed by the resulting systems.

In the context where the participants are involved in "traditional" business operations, the retailer and the producers are generally interested in keeping confidential the quotations they agree on about a specific selling. In particular, a producer may want to keep secret the quotation applied to a specific retailer. The retailer may want to make a private offer to a particular customer without showing the price for the same goods.

The second business context we consider pertains to a "fair trade" business model. In this case, it is probably of interest for all the participants to keep a certain level of transparency over the transactions they perform. In particular, a retailer operating in such a context should be interested in making publicly accessible the origin, and the price of the goods s/he sells. In this way, the customer can see exactly who the producer is, and if the price s/he is going to pay is somehow fair, and related to a reasonable treatment of the producer. In particular, the public nature of the information stored in the blockchain permits to analyse the items of the retailer's specific goods, and the prices applied over the time for the product, independently from the fact that the corresponding transactions belong to the specific choreography. This transparency will increment the retailer's reliability from the customer perspective in the specific business model context.

The retailing scenario provides an example of a multi-party business process that, when different operative domains are considered, does not differ much with the operative aspects, and the interactions put in place to reach specific objectives. Instead, the operative domains result in rather diverging needs when modalities of such interactions, and capabilities of successive analysis, are considered.

3. Multi-Party Business Processes in BPMN

In BPMN a multi-party business process can be represented using the choreography diagram. Such a diagram permits to express the interactions

among different parties, without revealing their internal behaviour. In a distributed environment, organisations wishing to collaborate can refer to specific choreographies that describe in detail how the different parties should interact to achieve common objectives. The integration of processes in this way leads to a more peer-to-peer collaboration, shifting responsibility for each execution step of the collaborative process to the individual nodes. Consequently, in a choreography approach, each participant is responsible for partial orchestration, based on its individual rules without a central coordinator, and the final behaviour is specified as a family of permitted message exchange sequence.

The most relevant elements used in choreography diagrams are depicted in Fig. 1. On the left, are represented the elements used for the business process's control flow, while on the right, the elements used for communication purpose. In general, a choreography model is composed of four different types of elements: events, gateways, sequence flow and tasks. **Events** can be a *start event*, representing the starting point of the process, and an *end event*, raised when the process terminates. **Gateways** act as either join nodes (merging incoming edges) or split nodes (forking into outgoing edges). We can have different types of gateways. A *parallel gateway* (AND) in join mode has to wait to be reached by all its incoming edges to be activated, and subsequently, in the split case all the outgoing edges are initiated simultaneously. An *exclusive gateway* (XOR) represents choices; it is initiated each time the gateway is reached in join mode, and it activates exactly one outgoing edge in split mode. For the *event-based gateway*, the outgoing branches activation depends on the reception of a message; these message events are in a race condition, where the first one that is triggered activates the branch

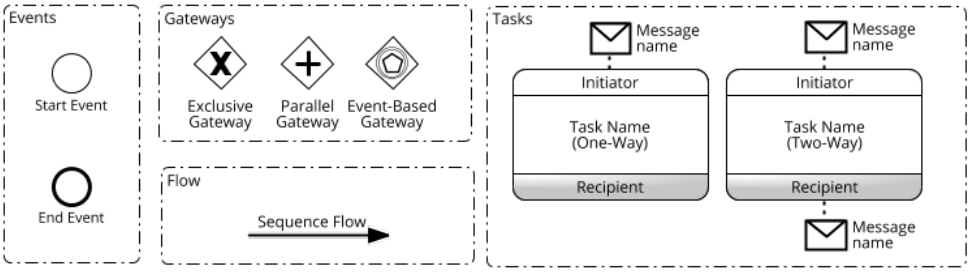

Figure 1: BPMN choreography elements.

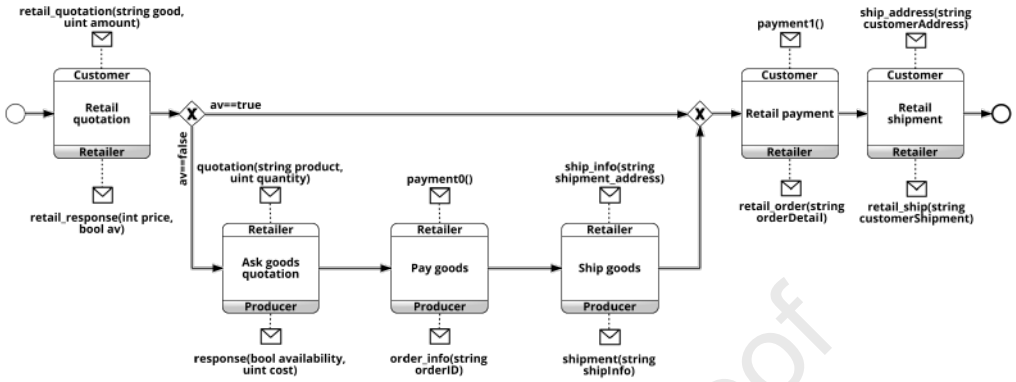

Figure 2: Retail process.

and disables the other ones. The **Sequence Flows** are edges used to connect all the choreography elements, permitting to specify the execution flow of the process. **Tasks** are used to define the message exchanges between two or more participants. They are represented as rectangles divided into three bands: the central one contains the task's name, while the others refer to the involved participants (the white band is the initiator, the grey one is the recipient). Messages can be sent either by one participant (One-Way tasks) or by both participants (Two-Way tasks).

Running example. The choreography reported in Fig. 2 represents the communications that should take place among the participants for the scenario described in the running example paragraph in Section 2. The model starts with the request by the customer for a quotation of goods. In case the goods are available, the customer proceeds with the payment, and the retailer commits to deliver the goods. In the other case, the retailer has to buy goods from the producer that the customer could have indicated, which then proposes a quotation. This quotation will be followed by the payment and the shipment of goods to the retailer that can close the customer's order with the final shipment.

4. The Multi-Chain Methodology

This section presents the proposed methodology supported by the Multi-Chain practical framework. The main steps of the methodology and the

involved actors and framework components are depicted in Figure 3, here shortly described.

The main steps of the methodology for a multi-party business process are as follows:

1. **Modelling**: the BPMN choreography model of the multi-party business process is produced by means of an appropriate modelling tool.
2. **Publishing**: the choreography model is stored in the models repository.
3. **Instantiation**: the choreography model is retrieved, the blockchain platform where deploying it is selected, and a corresponding choreography instance is created and uploaded into the pending instances repository.
4. **Subscription**: choreography participants subscribe to the instance to cover the required roles.
5. **Deployment**: when all roles are covered, a smart contract is automatically generated from the instance and deployed on the blockchain platform selected at step 3.
6. **Execution**: the subscribed participants interact via the blockchain in order to execute the choreography according to the behaviour specified at step 1.
7. **Auditing**: the logged choreography activities are inspected in order to check in a non-repudiable manner functional and quality-related aspects of the choreography execution.

The actors involved in the above methodological steps are as follows:

- **Choreography modeller**: the choreography modeller creates a choreography model so that interested participants can engage in multi-party business processes to reach specific objectives. The modeller can use any BPMN modelling tool supporting the design of choreography diagrams, and specific extensions needed to add the information required to deploy the choreography within a blockchain. Once the modelling activity terminates, the modeller can publish the model within a choreography model repository.

- **Choreography initiator**: the choreography initiator can be a person or an organisation that is interested in activating a choreography

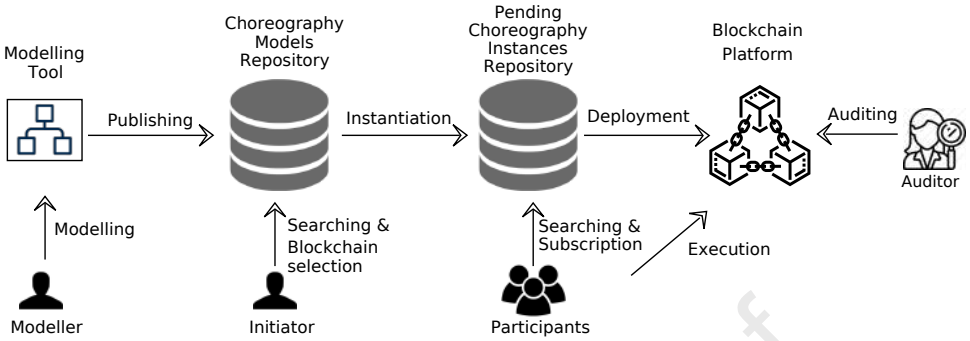

Figure 3: Multi-Chain supported methodology.

within a selected blockchain platform. The initiator searches and selects a choreography model among the ones stored in the repository to reach the objective. Once a model has been selected, the initiator will have to select the deployment blockchain environment according to his specific needs (see Section 2 for a discussion about the characteristics supported by different blockchain platforms). As a result, the initiator will publish on a separate repository the instantiated choreography, so that interested participants can subscribe to one of the foreseen roles.

- **Choreography participant**: a choreography participant is generally constituted by an organisation that is interested in taking part in a choreography execution to reach specific objectives. To do so, it searches for pending choreography instances, and it subscribes to those in which it can fruitfully play a role. Once all the mandatory roles result to be covered, the choreography is deployed within the selected blockchain infrastructure by defining suitable smart contracts. Successively all the participants will interact to perform the business process defined by the choreography.

- **Choreography auditor**: an auditor accesses the blockchain to analyse the stored data and to check that the transactions corresponding to a specific choreography execution are in line with the expected ones. In some case, the auditor can be impersonated by any participant, depending on the adopted blockchain technology [6].

The relevant software components that form the tool-chain supporting the methodology are as follows:

- **Modelling tool**: a modelling tool makes it possible to represent a choreography diagram and augment it with specific blockchain-related information (e.g., the structure of messages).
- **Choreography models repository**: it permits the storage of choreography models and their retrieval. A specific component provides an interface to the initiators to search choreographies and select the blockchain platform. This component will also take care of properly instantiating the choreography and uploading it into a repository where choreography instances in a pending state are stored.
- **Pending choreography instances repository**: it stores pending instances and allows choreography participants to subscribe to them. This can be a separate repository or a partition of the previous one.
- **Translator**: a transformer is needed to automatically derive smart contracts from a choreography instance for the selected blockchain infrastructure.
- **Blockchain platform**: a blockchain platform is needed to support the choreography's execution and possible related auditing activities.

It is worth notice that some of the steps described above can be implemented with different level of complexity and automatic support. In particular, search mechanisms made available by the repository can go from simple pattern matching on specific choreography metadata to more semantically related aspects. The selection of participants who take part in a choreography can go from a closed list strategy where the initiator select "off-line" the participants, and gives them a reference to the pending instance to join, to more complex mechanisms where requests for participation are issued through the system itself. Notably, a choreography model can include two different kinds of roles. Mandatory roles for which at least one participant has to subscribe, and optional roles for which a participant can join after the choreography has already started. Multi-Chain covers all the functionality presented here, except for auditing, for which it resorts to the mechanisms made available by each specific blockchain technology. Simultaneously, the focus has been put on the smart contract's derivation using a model-driven engineering approach, whose details will be provided in the next section. In contrast, automation for the other activities has been kept at a basic level.

Running example. Considering the retailing scenario, we can imagine that a modeller inserts the model reported in Figure 2 on a repository. At this point, the initiator's role can be probably played by the customer that willing to buy a given good selects a choreography that permits him to reach such a goal. As part of the process, the customer also defines the characteristics of the process s/he would like to be involved in. As a result, this will lead to selecting a specific blockchain infrastructure, satisfying the specified properties. Then either the customer directly invites a retailer, or the platform notify a possible retailer, anyway when s/he will join the choreography can be activated generating and deploying the needed smart contracts on the selected blockchain technology. During the execution phase, it will be possible that the retailer will invite a producer to join the choreography so as to proceed to the provisioning. Depending on the selected infrastructure, different levels of transparency and auditing mechanisms can be accessed by any participant to check the choreography status and the contents of the exchanged messages. At the same time, the correct order will be guaranteed by the blockchain itself.

5. Multi-Chain: blockchain infrastructures generation

In this section we describe how the blockchain infrastructures are derived by means of the Multi-Chain model-driven approach. We focus in particular on the translation from model to code, i.e., the contracts creation.

5.1. Ethereum vs Hyperledger Fabric infrastructures

Ethereum and Hyperledger Fabric have been conceived with rather different objectives and usage scenarios. Therefore, it is not surprising that the transformation from a choreography diagram to the blockchain infrastructure to be deployed over the specific technology is different. In particular, in Ethereum the instantiation of a choreography diagram leads to the generation of a single smart contract to be deployed on the public network. This contract explicitly includes the users' addresses that subscribed to the roles in the previous phase. In such a way once deployed the smart contract can be used only by the subscribed users, and every functionality is then enforced both considering the order of the operations and the roles, as specified in the choreography model.

In Fabric, the situation is rather different since there is no global network and so, for each deployment, it is necessary to create not only the smart

contract (chaincode) but also the network infrastructure. In particular, in our approach, any choreography instance is represented by an Hyperledger channel, and each role is associated with a unique Fabric organisation. At this point, any user subscribed to a specific role becomes a member of the organisation representing that role in a specific model. Technically, the user will be associated with the organisation through an identity released exploiting cryptographic artefacts. Each channel is composed of (i) the chaincode representing the choreography instance behaviour and (ii) organisations, representing instance participants, communicating through a channel. To identify specific users, an attribute-based access control strategy is adopted. This encodes an attribute representing each user member's identity in the organisation, and this will be used to restrict the visibility of data on the deployed chaincode. This mechanism guarantees the privacy of exchanged information between different users covering the same role on two different instances that are both included in the same organisation. This guarantee that each user can see only data related to the contracts in which s/he is directly involved.

The methodology introduced in Section 4 is practically realised in the Multi-Chain framework by relying on the two blockchain platforms discussed above. Once the modeller has created a choreography model and published it in the Multi-Chain repository, an automatic procedure instantiates, following the model specification, the consortium and the organisations involved in the Fabric network. Then, the initiator user creates an instance choosing between Ethereum or Fabric according to her/his needs. In the Fabric case, every time a participant user subscribes to a role, the corresponding identity is created and added to the private network. When all the roles of the instance are fully covered, the deployment phase can start. Here the translator, according to the instance type (Ethereum or Fabric), generates the specific smart contract and deploys it in the respective network. Finally, the execution of the instance can start, allowing the participant users to interact with the blockchain, thus advancing the state of the contract.

5.2. Multi-Chain translator

We describe here the technical differences between creating an Ethereum contract and a Fabric one by showing the relative examples using the Retail process scenario described in Section 2. We then describe how a model-driven approach permits to support the methodology described in Section 4. The smart contract generation is an automatic phase where the choreography instance is translated into code. The generation of the code starts after the

parsing of the choreography model, performed using the Camunda library[3], properly extended by us to deal with the choreography diagrams syntax as defined in the standard. We describe the generated code both for Ethereum and Fabric, using respectively Solidity and Javascript. The logic behind the code generation for the control flow elements is similar for both technologies. In particular, a generated smart contract permits the participants to interact according to the corresponding choreography protocol. To do so the generation foresees the introduction of specific methods for each message exchange to be performed and the introduction of mechanisms to track the status of the protocol so as to enable the various message exchanges in line with the specification. Instead, differences are introduced concerning the derivation of supporting mechanisms for the properties listed in Section 2. In particular, in Fabric, the introduction of mechanisms to support confidentiality, that are not present in Ethereum, asks to derive a complex transformation procedure for the definition of specific users rights and their control, and the introduction of a *public* and *private* state for the transactions.

Listing 1 and Listing 2 show the template for the header respectively for the smart contract in Ethereum, written in Solidity, and the header of the chaincode for Hyperledger Fabric (ChoreographyPrivateDataContract), in JavaScript. In the last case, it can be noted the introduction of two utility classes: (i) ChorographyState and (ii) ChoreographyPrivateState.

```
contract RetailProcess{

  enum State {DISABLED, ENABLED, DONE} State s;
  struct Element{string ID; State status;}
  struct StateMemory{
  string good;
    uint amount;
    uint price;
    string shipment_address;
    ...}

  event functionDone(string eventID);
  Element[] choreographyElements;
  StateMemory currentMemory;

  mapping (string=>uint) position;
 string[] elementsID = ["StartEvent_102vawy", "Message_0b917rc", "
     ExclusiveGateway_042aut8", "Message_ID", ...];

  mapping(string=>address payable) roles;
  mapping(string=>address payable) optionalRoles;
```

[3]https://docs.camunda.org/javadoc/camunda-bpm-platform/7.11/

```
21
22    string[] roleList = [ "Retailer", "Customer" ];
23    string[] optionalList = ["Producer" ];
```

Listing 1: Ethereum: Contract header

```
1  const chorID = '68e81c58-2ca9-4a92-b438-76f06f358fa3'
2  const contractName = 'contracte3158a2b-40b7-43b0-9ae2-d19dacb39839'
3  const Status = { DISABLED: 'disabled', ENABLED: 'enabled', DONE: 'done'
       };
4  const chorElements = ["StartEvent_102vawy", "ExclusiveGateway_042aut8",
       "Message_0b917rc", ...]
5  const roles = { Customer: '0rg1MSP5fe1cdac280183175ccb152e', Retailer: '
       0rg2MSP5fe1cdac280183175ccb152e', Producer: '
       0rg3MSP5fe1cdac280183175ccb152e'}
6  const collectionsPrivate = {CustomerRetailer: 'collection' + roles.
       Customer + roles.Retailer, ...}
7  const subscriptions = { Customer: '5fe0b7aa2801833b2c91a2d3', Retailer:
       '5fe0b82f2801833b2c91a2e1', Producer: '5fe1ce56280183175ccb153a'}
8
9  class ChoreographyPrivateDataContract extends Contract {
10     constructor() {
11         super(contractName)
12     }
13
14 }
```

Listing 2: Hyperledger: ChoreographyPrivateDataContract class

In Solidity (Listing 1), a contract keeps track of the choreography instance state through the list of elements `choreographyElements` (line 13) and the structure of variables `currentMemory` (line 14) containing all the information influencing the state of the contract. Each element of the former list is a structure of type `Element` (line 4) representing the information related to that model element (i.e., its identifier and current status), while the current memory has type `StateMemory` (line 5) and it contains all the global variables appearing in the model. The states of an element, defined by the enumeration `State` (line 3), are as follows: `DISABLED` is used when the element has never been called and is waiting for being enabled, `ENABLED` when is waiting for being executed, and `DONE` once it has completed the execution. The event `functionDone` (line 11) is emitted for each completed element, and it permits to retrieve the transactions of the contract directly, so to improve the performance of the auditing phase. Also, the function is used to notify the partners about a possible contract state change. The header also includes the choreography elements list, with their identifiers `elementsID` (line 17), and the list `roleList` and `optionalList` of the mandatory and the optional roles involved in the choreography (lines 22-23).

In Fabric (Listing 2), the contract is defined through the `Choreography-PrivateDataContract` class. Like Ethereum, the class keeps tracking each element of the choreography within an associative object. Also, in this case, the element life-cycle is composed of three states, listed in the `Status` object (line 3). The `chorElements` (line 4) object maps choreography elements to their individual state, and it is included in the ledger state. Additional information like the contract name and the id of the choreography are reported in lines 1-2. The roles declared in line 5 maps choreography roles to the Fabric Membership Service Providers belonging to the related organisations to guarantee confidentiality. In line 6, the definition of the private collections is done coupling all the different roles of the model inside the `collectionsPrivate` object. Also, in line 7 the roles are associated with the subscribed users, which identities were previously created inside the organisations.

In the two contracts just after the header, an access control function is introduced to define the participants to a message exchange. In the Ethereum case this is done using the `modifier` statement, and in the Hyperledger case using an identity check.

```
24  modifier checkMand(string memory role) {
25    require(msg.sender == roles[role]);
26    _;
27  }
28  modifier checkOpt(string memory role) {
29    require(msg.sender == optionalRoles[role]);
30    _;
31  }
```

Listing 3: Ethereum: Modifiers.

In particular, Listing 3 reports the modifiers `checkMand` (lines 24-27) and `checkOpt`(lines 28-31). They check if the mandatory/optional role of the sender in that particular function corresponds to the role for which the same account was subscribed. These constructs are used to enforce, from the contract side, the right identity of the sender according to what expressly defined in the choreography instance.

```
15  roles.Customer === ctx.stub.getCreator().mspid && ctx.clientIdentity.
        assertAttributeValue('role', subscriptions.Customer)
```

Listing 4: Hyperledger: Enforcing controls

The same control is quite different in Fabric (listing 4), indeed two main controls are necessary before executing a message. One controls the organisation and one the user's attribute. The first part of the control checks if the caller is a member of the organisation having the rights for executing

the function. This is done by checking the MSP of the transaction creator, respect to the `roles` list defined in the contract header. The identity of the caller is then checked through the encoded attribute defined in the identity certificate. The identity certificate allows distinguishing not only the organisations corresponding to the right role but also the specific user, previously associated with a single role. In this way, the caller attribute is first retrieved and then compared to the id of the right role inside the `subscriptions` object.

The smart contracts generation continues by appending the functions corresponding to the translation of the elements included in the choreography model for both technologies. The concept of choreography *task* is concealed in favour of the connected messages. In particular, a one-way choreography task is represented by its message, and similarly, the two-way task is represented by its two messages. Thus, the choreography elements appearing in the contract can be divided into two main categories: *messages*, representing the interactions between participants, and *control flow* elements representing the logic of execution.

In Ethereum, the translation generates a public function for each message and a private function for each element of the choreography control flow. In Hyperledger Fabric instead, both messages and control flow elements are represented with an *async* function.

Listing 5 shows the Solidity public function depicting a message exchanged between two participants in a choreography task. The function name in the contract is represented by the message identifier inherited from the model, while the parameters from the name of the message.

```
32 function Message_1wrru53(string shipment_address) public
33                 checkMand(roleList[0]) {
34   //checking the status of the current element that is the invoked
         message
35   require(elements[position["Message_1wrru53"]].status
36           ==State.ENABLED);
37   currentMemory.shipment_address=shipment_address;
38   done("Message_1wrru53");
39   //it enables the next element, in this case another message
40   enable("Message_1tq0g6g");
41 }
```

Listing 5: Ethereum: A message function.

The modifier `checkMand` is called with the assigned role (line 33). Once the right identity of the caller inside the function is ascertained, a second check on the enabled status of the task is performed (lines 35-36). After that, in

the body of the function, the `shipment_address` parameter is stored in the memory of the contract (line 37). At this point, the status of the current element is changed to `DONE` (line 38) and the successive one is set to `ENABLED` (line 40).

Listing 6 shows the implementation of a message in Fabric. In line 17 the actual public state of the choreography is retrieved from the external utility class `ChoreographyState` and it is used for the next operations. Line 18 reports the controls performed before allowing the execution of the message. In the condition of the conditional statement, we have the check of the status of the actual message, identified by its `Message_id`. This permits to enforce the execution of the right sequence of functions. The other two expressions in the condition are related to the check of the right user and organisation as described in listing 4. If the user is the right one, the private state associated to him is recovered (line 19), calling the external class `ChoreographyPrivateState`. This state concerns the participants' interaction in which only the information changed between them is stored. In particular, to get the private state some information must be passed: (i) the Fabric ctx, (ii) the private collection between the sender and the receiver of the message and (iii) the id of the choreography, automatically set by the translator in the generation phase. Like in Ethereum, the actual message is set as *Done* and the next one is *enabled* (line 20-21). Finally, the public and the private state of the choreography are updated (line 22-23). In particular, these operations are done without passing directly the information inserted by the user, but exploiting information stored in the context object (ctx). Indeed, the ctx encapsulates the transient data that are then extracted in the invoked functions, in this way, are not explicitly visible in these operations.

```
16  async Message_0b917rc(ctx) {
17     const choreography = await ChoreographyState.getState(ctx, chorID)
18     if(choreography.elements.Message_0b917rc === Status.ENABLED && roles
           .Customer=== ctx.stub.getCreator().mspid && ctx.clientIdentity.
           assertAttributeValue('role', subscriptions.Customer)) {
19         const choreographyPrivate = await ChoreographyPrivateState.
               getPrivateState(ctx, collectionsPrivate.CustomerRetailer,
               chorID)
20         choreography.setDone('Message_0b917rc')
21         choreography.setEnable('Message_1xxdwx2')
22         await choreographyPrivate.updatePrivateState(ctx,
               collectionsPrivate.CustomerRetailer)
23         await choreography.updateState(ctx)
24         return { choreography, choreographyPrivate }
25     } else {
26         throw new Error('Element Message_0b917rc is not ENABLED or
               submitter not allowed, only the Customer can send this
```

```
                transaction')
27      }
28  }
```

Listing 6: Hyperledger: Message Function

Also in the case of gateways, we have a similar implementation. Here below, we only show the *exclusive gateway* implementation for both technologies. As it can be imagined for all the other gateways, we have a similar structure. The logic of the gateway for the Ethereum case is depicted in Listing 7. Here, the next element is enabled only after the evaluation of a condition that discriminates which element to enable. The condition managing the choice in the conditional statement (line 45) is inherited directly from the outgoing sequence flows of the exclusive gateway represented in the Choreography model. The if-else control (lines 45-50) in the smart contract guarantees the mutual exclusion in the evolution of the control flow, limiting the execution to the first satisfied condition.

```
42  function ExclusiveGateway_042aut8() private {
43    require(elements[position["ExclusiveGateway_042aut8"]].status
44            ==State.ENABLED);
45    if(currentMemory.availability==false){
46      enable("Message_1h3ew61");
47      Next_Element_ID();
48    }else if(currentMemory.availability==true){
49      enable("ExclusiveGateway_1johog7");
50    }
51    done("ExclusiveGateway_042aut8");
52  }
```

Listing 7: Ethereum: Exclusive Gateway Function.

A similar structure is used in the Fabric contract (Listing 8). Firstly the status of the actually invoked gateway identified by `Gateway_id` is checked (line 30). Then its status is set to done (line 31), and the evaluation of the variable is performed. Depending on its value, the function enables the next element to be a message or a gateway, identified by its id. At this point if the next element is a message, the public state is updated calling the external function `updateState` that will insert the new status of the elements (line 34); otherwise, it is directly called (line 37), and the public state is updated in the next functions.

```
29  async ExclusiveGateway_042aut8(ctx, choreography, choreographyPrivate) {
30      if(choreography.elements.ExclusiveGateway_042aut8 === Status.ENABLED
            ) {
31          choreography.setDone('ExclusiveGateway_042aut8')
32          if(choreographyPrivate.av==false) {
33              choreography.setEnable('Message_1h3ew61')
```

```
34                 await choreography.updateState(ctx)
35             } else if(choreographyPrivate.av==true) {
36                 choreography.setEnable('ExclusiveGateway_1johog7')
37                 await this.ExclusiveGateway_1johog7(ctx, choreography,
                       choreographyPrivate)
38             }
39         } else {
40             throw new Error('ExclusiveGateway_042aut8 not ENABLED')
41         }
42     }
```

Listing 8: Hyperledger: Exclusivegateway implementation

6. Implementation of Multi-Chain Practical Framework

In this section, we describe the implementation of the Multi-Chain tool that reflects the concrete implementation of the steps described in our methodology. The reader can practically experiment the framework deployed at `http://virtualpros.unicam.it:8080/MultiChain/`. This as been implemented using the Rinkeby-Ethereum Testnet[4], which is a sandbox copy of the Ethereum blockchain. For Fabric instead, the network is created dynamically during the deployment phase according to the process structure described in Fig. 3.

6.1. Modelling

The modelling phase is the starting point of the choreography life-cycle. To support it, we have integrated a modelling environment into the framework as showed in Fig. 4. The modelling area offers several functionalities, such as the creation, the import, the export and the saving of a model in the Multi-Chain repository. This choice permits to avoid the common interoperability issue affecting BPMN modelling environments, thus guaranteeing the produced choreography's full compatibility with all the features provided by the rest of the framework.

Due to its intended abstraction level, a choreography model does not include enough details to enable an automatic generation of code directly. For this reason, we extended the modelling environment so to ask to the modeller additional data about *(i)* messages and *(ii)* guards. This data are needed to permit the deployment in a blockchain infrastructure. However, they are included without differentiating Ethereum from Fabric.

[4] `https://www.rinkeby.io/`

Therefore, during this phase, the modeller has to annotate each choreography task's message(s) with the parameters needed to perform the underlying function call in the generated smart contract. To facilitate this procedure, the specification of a task is supported by an intuitive panel that requires the insertion of the following information:

- the participant names
- the names of the exchanged messages
- the parameters
- a specific indication if the message includes a payment (supported only by Ethereum, and ignored for the case of Fabric).

The result of this procedure is the addition of a list of parameters after the message name in the form of:

$msgName(paramType_1\ paramName_1, \ldots, paramType_n\ paramName_n)$.

The Ethereum blockchain natively supports financial transactions among the interacting partners for exchanging specific amounts of cryptocurrency. Therefore, the modeller has the possibility to include messages in a choreography that can produce financial transactions. In case the *payment* checkbox is selected for a given choreography message, the corresponding function is created, and the message is automatically filled with no parameters. The name assigned to the message is of the form $payment\,n\,()$, where n corresponds to a counter that is incremented for each new payment function added to the model, to obtain unique names. The lack of any parameter is justified because the only information required by the payment function refers to the involved participants, which can be directly and automatically retrieved from the task description. The amount to be paid will be indicated by the sender during the choreography execution, exploiting the dedicated page. The resulting transaction will transfer the amount in Ether from the sender to the receiver wallet. To notice, the payment functionality is enabled only on Ethereum, due to the absence of a native cryptocurrency in Fabric.

Another fundamental aspect to consider when executing a choreography is related to the guards of the involved exclusive gateways. Each sequence flow outgoing from an exclusive gateway must refer to a boolean expression that indicates the path to be triggered. This expression could be written using the standard comparison operators for boolean, numeric and string variables.

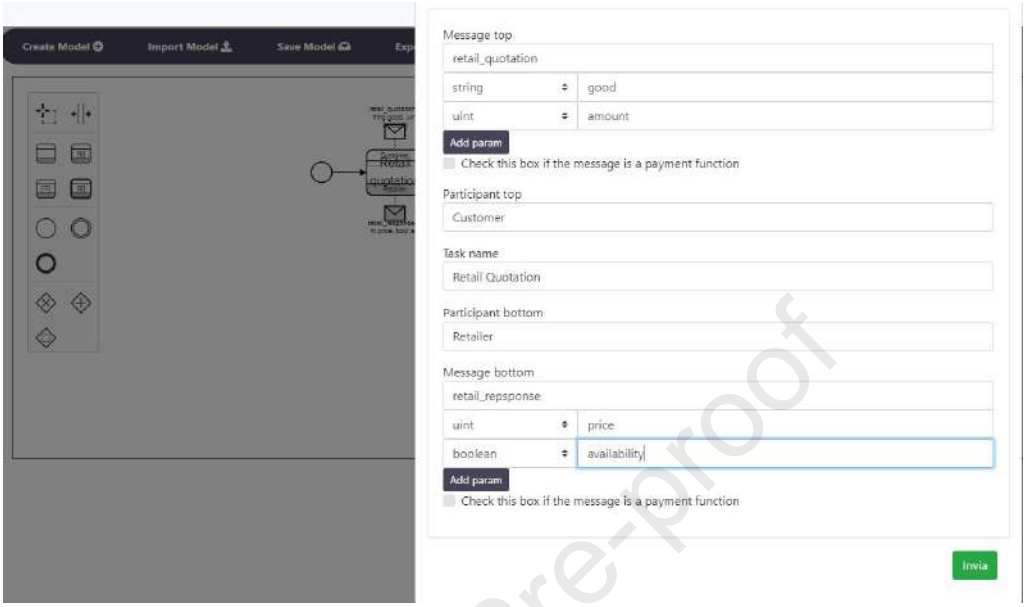

Figure 4: Multi-Chain modeler.

After a model is created, it is possible to save it storing the choreography file inside the repository. This operation will also generate new unique organisations associated with the participants that can be used later in the Fabric subscription and the deployment phase.

6.2. Publishing, Instantiation and Subscription

Here we describe the multiple blockchains support for publishing and instantiating a choreography specification. In particular, after the instantiation phase, the distinction between the two resulting artefacts to be deployed on a specific blockchain is evident, while till the publishing phase, the model is unique.

Once a choreography is published into the Multi-Chain repository, it can be accessed via an intuitive user interface. However, to interact with the repository, it is necessary to register and login into the platform. These operations require a name and a password to create the user's identity inside the platform. However these are only preliminary high-level credentials; for identifying the user inside the blockchain processes, an Ethereum address or a Fabric organisation will be assigned later. After the login, the user is redirected to the homepage depicted in Figure 5 , which shows the uploaded

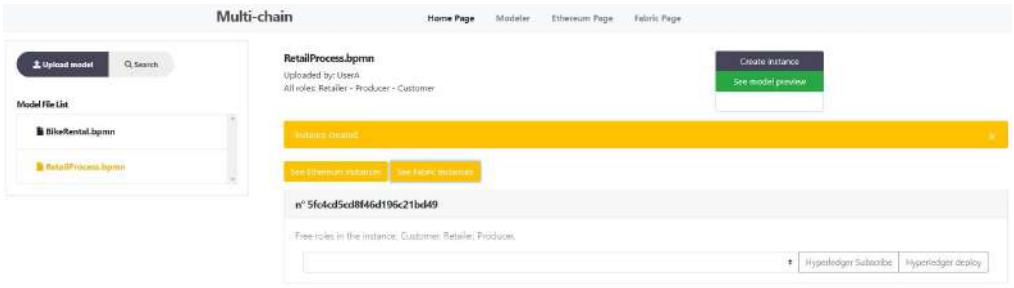

Figure 5: Tool homepage with focus on the Fabric instances.

and instantiated Retail process example.

On the left side of the web page, the user has the possibility to publish a new model, by uploading the corresponding file. Alternatively, s/he can search for an existing one.

The searching phase is an important aspect of the framework as it enables reusability and facilitates the meeting between supply and demand of services. Once logged in, any registered user can search for a particular choreography, and the framework proposes the list of all models matching the searched topic. These are listed below the search form.

The information about the selected choreography is shown on the right side of the homepage, depending on the instance topology (Ethereum or Fabric) the platform shows different information. In particular, common information like the model owner, the maximum number of involved participants and the required roles are shown. Also, the preview of the graphical representation and the possibility of creating a new choreography contract are available.

A model instantiation results are two choreography instances created for the two implementations one for Ethereum and one for Fabric. They are kept in a "suspended" state while waiting that all the mandatory roles are subscribed. Fig. 5 shows the home page with the retail process instantiated, in particular, the fabric instance is highlighted.

Before deploying one of the two possible instances, the choreography participants must be filled by the users during the subscription phase. For Ethereum, when the user subscribes a role, it's necessary to associate her/his Ethereum address through the Metamask browser plugin that manages blockchain accounts. At this point, the role is considered covered and it will be associated in the blockchain and the Solidity contract to the user address.

For Fabric, the procedure is quite different since the user's identity is directly created after the subscription. Indeed, as described in Section 4, roles are associated with Fabric organisations. These automatically generate later the artefacts for the user's identity that become a member of the organisation covering that role in that specific instance. Thus, the interface is necessary only to select the desired role, without the need for additional operations.

When one of the two choreography instances has no more vacant mandatory roles, the partnership is complete, and the smart contracts generation phase can start, deploying it on the chosen blockchain. For Ethereum, if the contract has some optional roles, the subscription form remains enabled in the homepage with only the optional roles, also after its deployment. In case a user selects an optional role, the correlated subscription function will be triggered directly on the already deployed smart contract. This operation generates a standard transaction that needs to be accepted via the Metamask plugin. The details regarding the smart contract generation were described in Section 5.

6.3. Deployment

Once the contract has been generated, the framework automatically deploys it into the selected blockchain. Depending on the chosen technology, the deploy operation will be different. For Ethereum, the server will generate a transaction that deploys the generated Solidity contract on the blockchain. For Fabric the procedure is more complicated since, for each new instance, a new channel must be created. This happens in the back-end, where the system automatically generates the artefacts related to organisations, orderers, channels and chaincodes. In particular, organisations artefacts are produced after the model publishing, users' identities in the subscription phase, chaincode and the channels in the deployment.

6.4. Execution

Once a new contract is deployed into the blockchain, the execution phase takes place, and the participants can collaborate using the functions exposed by the contract. To facilitate these interactions, there are two execution pages accessible by each participant. These pages enable the interaction with Ethereum contracts or with the Fabric ones.

Figure 6 shows the Fabric page concerning the deployed Retail process example. However, this execution page has some common characteristics with the Ethereum one. On the left-hand side, the interface reports a list

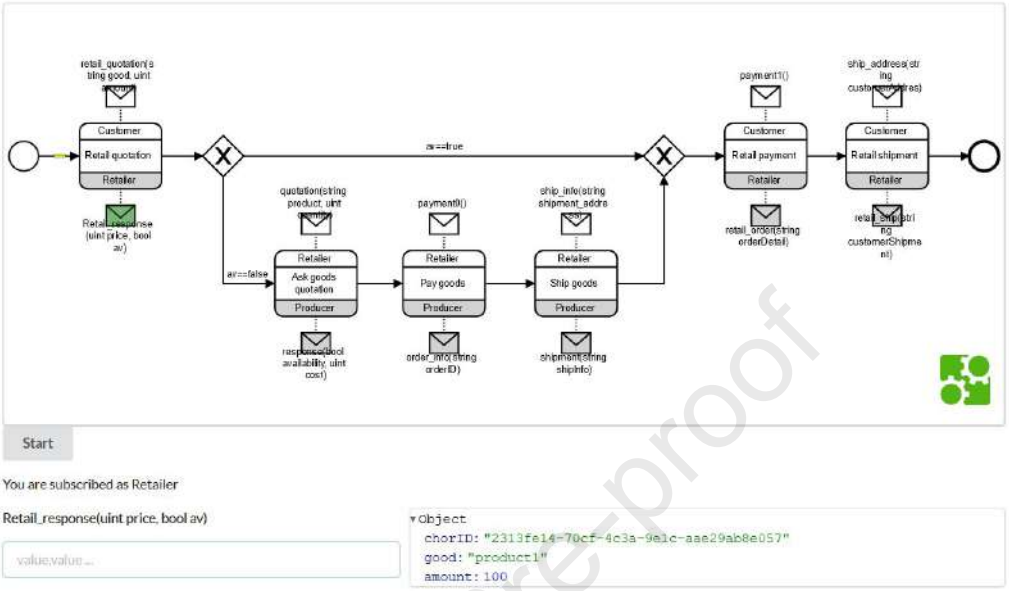

Figure 6: Fabric execution page.

of all contracts to which the participant is subscribed. In the right-hand side, a preview of the model is shown: in green, the messages completed are indicated, and the ones actually active. For these, the window also includes the forms that are dynamically constructed by the tool. Each form contains many information, like the name of the message, the role of the participant, the space for inserting all the required parameters, and the submit button. Notably, the submission form is visible only to the participant in charge of sending the enabled message.

By double-clicking on a completed message, a little panel with the exchanged values is shown. However, for Ethereum all data will be visible. In Fabric instead, since the main requirement is the privacy of exchanged information, there will be visible only to the participants. When an Ethereum message is sent, the transaction has to be confirmed using the Metamask pop-up. It contains the gas price plus the total amount of Ether to spend for the transaction. As soon as the transaction is included in a block (i.e., it is mined), the related event is emitted. The front-end uses this event to update the interfaces of all participants involved in the choreography with the new contract status, thus enabling the next admitted message(s). In Fabric instead, a function execution does not require the payment of any fee,

making the transaction process faster. It is worth noticing that the choreography is executed in a distributed manner, since the participants interact via the front-end directly with the blockchain, without referring anymore to the back-end component.

7. Discussion

In this work, we presented a multiple blockchain technologies implementation supporting the full life-cycle of choreography diagrams. In particular, the model-driven approach allows specifying the high-level behaviour of distributed systems, just focusing on their messages exchange. These models are then deployed and executed inside the blockchain, guaranteeing a trusted communication also in untrusted contexts, with an immutable proof of the executed communication. In particular, we chose to adopt both the permissionless Ethereum and the permissioned Hyperledger Fabric blockchains. Indeed, their different nature allows covering a large set of properties as highlighted in Section 2.

It is worth mentioning that in a permissionless blockchain, it is possible to use encryption to obtain privacy restrictions, and have similar benefits concerning the ones provided by a permissioned blockchain. However, in a permissionless blockchain, the encrypted data are saved in each node of the network. Then this information can be used by a malicious node that with enough time and computational resources could be able to broke the encryption and get access to sensible data. Instead in a permissioned blockchain like Fabric, this situation is prevented by the technology that limits the distribution of confidential data exclusively to authorised nodes via channels and private data collections. However once considered relevant, a specific profile to support encryption on a permissionless blockchain could be added as an additional option to the Multi-Chain infrastructure.

An example of data confidentiality in Multi-Chain is reported in Fig. 7 where the execution of the Retail process (described in Section 2) is performed. The figure shows the Producer perspective inside the Fabric execution page. Here is possible to notice that the currently active message is the *retail_response* so this means that the previous one was already sent with its information. However, the current user is the Producer so s/he is not allowed to see what the Customer and the Retailer are sending. Indeed, the Producer visibility is restricted since in this case the interactions are private, so a participant will have visibility only on data directly sent or received.

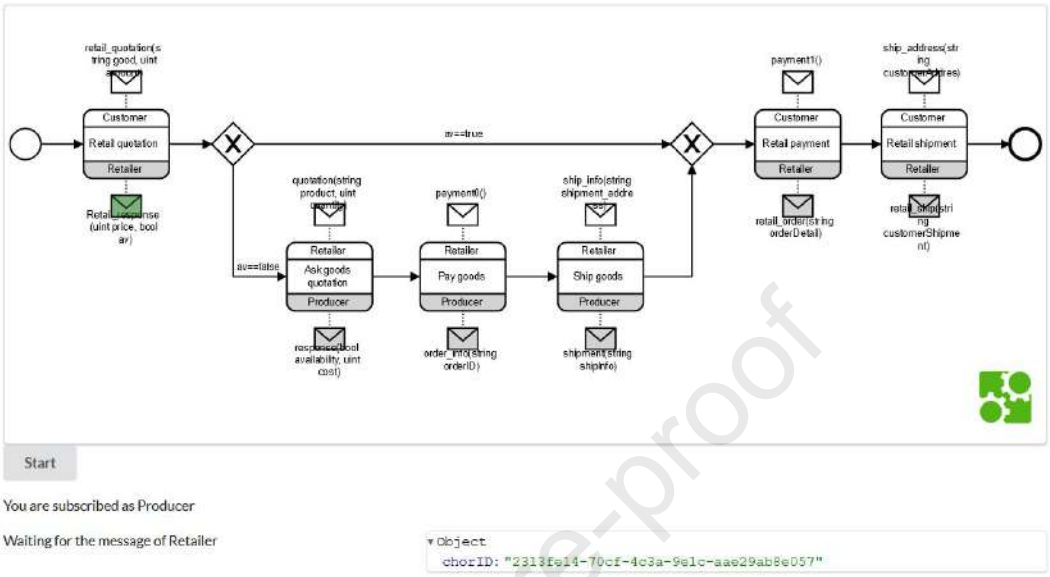

Figure 7: Execution of the retail process over the Fabric blockchain.

In Ethereum instead, the same deployed process provides a transparent view of the messages. Fig. 8 shows always the Producer participant but this time s/he can see the request made by all other participants.

A similar discussion could arise considering the identity of the participants in a private blockchain. Indeed creating a private Ethereum network is possible to have restricted visibility of information shared only within the participants of the private network. From one side this could partially solve the data confidentiality, but from the other side, it is a really rigid structure not adequate for dynamic systems since a dedicated network should be created from scratch. In Hyperledger Fabric the network is completely configurable and composed of elements able to manage the identities (certification authority) and control the access policies (membership service provider) in an automatic way. This permits to have a dynamic network, that is configurable both at design time and run-time. In the Multi-Chain approach, the Fabric network is dynamic and updated instantaneously each time that a choreography instance is created.

Another aspect to consider in this discussion is related to the choice of the best blockchain implementation to use in the Multi-Chain approach. We discerned the two approaches highlighting mainly the privacy and con-

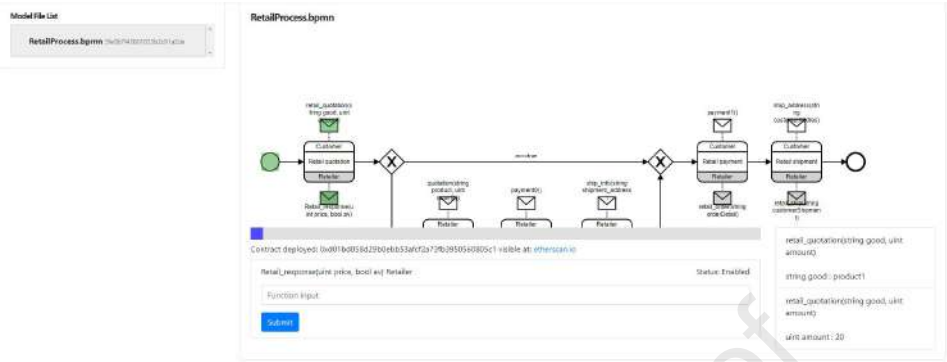

Figure 8: Execution of the retail process over the Ethereum blockchain.

fidentiality aspects without considering too much the different throughput between permissioned and permissionless blockchain. Fabric is more recommended in scenarios where a high number of transactions per second (TPS) are required since it can reach the moment we are writing 20000 TPS and instead Ethereum just 20 TPS. At the same time Fabric is not able to guarantee audibility on the large since the communication between participants is restricted using channels. For solving such contrasting requirements we planned to improve Multi-Chain including in the model-driven the possibility to specify the policies governing confidentiality and privacy of data. This will permit to Multi-Chain to customise the network during the generation phase according to the user requirements.

7.1. Performance analysis

We report here the results of the experiments we made on Multi-Chain to assess its performances and the costs of the approach concerning the translation, deployment, transaction execution and, for Fabric only, the network creation.

Fig. 9 compares the times that the translator needs to generate ten smart contracts, for each of the two blockchains, derived from the running example model. The average time for the translation is 22 ms for creating Ethereum contracts and 24 ms for creating the Fabric ones. The differences are minimal and the two platforms can be considered equivalent during this phase. Fig. 10 represents, instead, the times taken for the deployment of the same contracts previously created. For Ethereum, the trend is not constant, but on average it takes around 17 s. For Fabric instead, the trend is more uniform, but the requested time is higher, as it takes 79 s on average. This

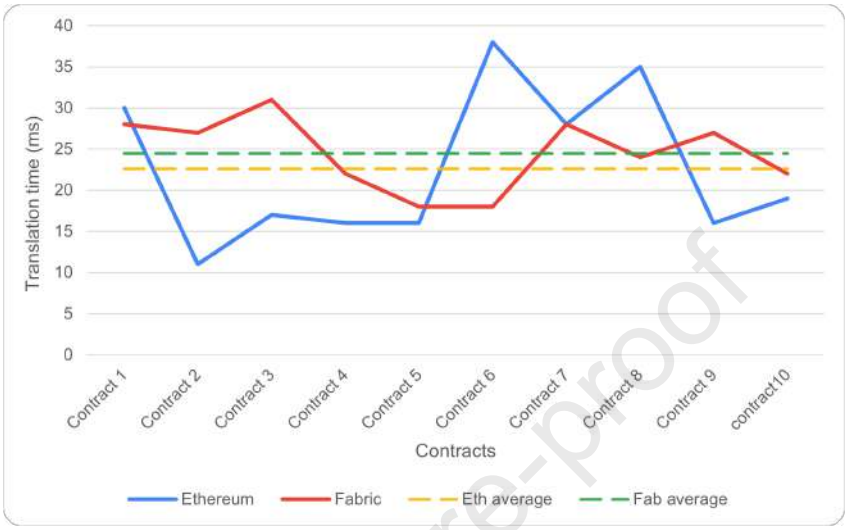

Figure 9: Translation time of 10 running example choreography instances.

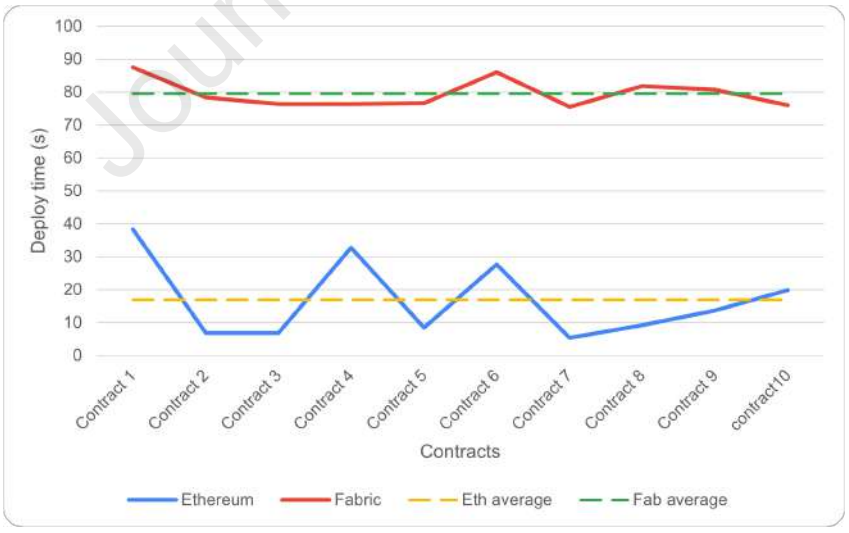

Figure 10: Deploy time of 10 running example choreography instances.

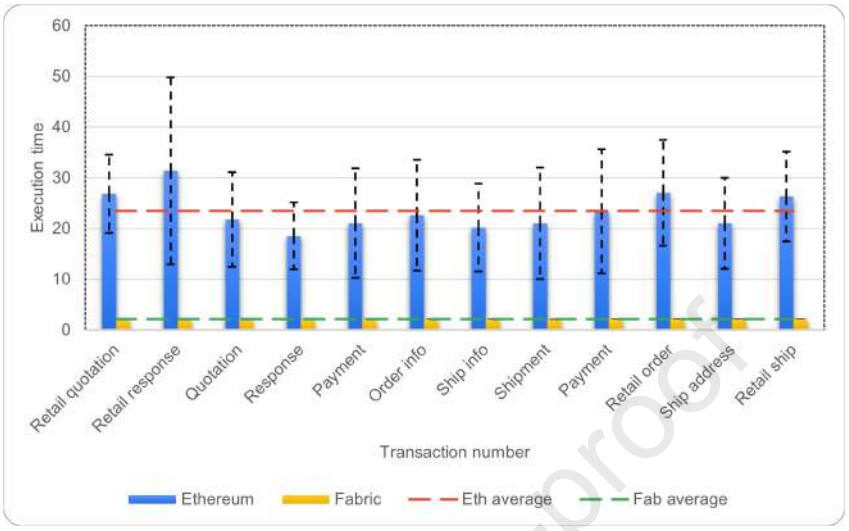

Figure 11: Average transactions execution time for the running example.

degradation of performance is motivated by the necessity for Fabric to approve and verify a sequence of stages in each peer of the network. In Fig. 11 we compare the average time required for executing a specific transaction of the running example. Each transaction was executed 10 times for each blockchain technology using the different smart contracts derived from the running example model. Here the differences between the two technologies are more evident. In Ethereum, the time necessary for processing a transaction fluctuates between 14 s and 38 s on average, with a significant standard deviation moving between 4 s and 19 s. However, on average a generic transaction is executed in 23 s for a single execution; this result is in line with the standard performance of the Rinkeby network for the inclusion of a transaction. In Fabric, such a trend is much more regular and we do not have significant differences between distinct transactions. In general, the average time necessary corresponds to 2 s with a standard deviation that is not relevant. Finally, in Fig. 12 we report the performance for the network creation. This measure is reported only for Fabric since there is not the same need on Ethereum. Fabric is based on a private network and, therefore, for each new choreography, it is necessary to create a new preconfigured network considering the involved organisations and successive participants' identities. In the experiments, we isolate 10 different networks instantiation and we observe on average a generation time of 20 s. This additional time should be considered

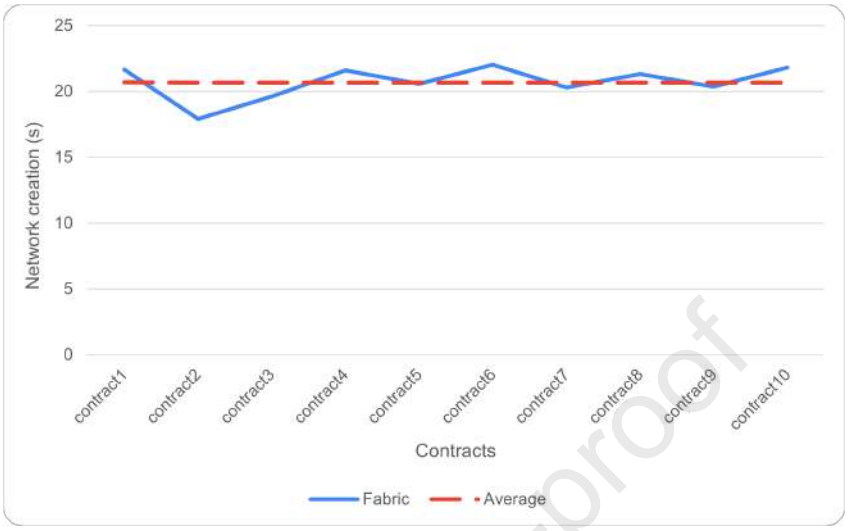

Figure 12: Time required for creating a Fabric network.

one-time only during the first instantiation of the choreography.

Another interesting analysis to evaluate the effectiveness of Multi-Chain is the cost for the execution. In this respect, we report only the test on the Ethereum technology, since it allows to measure the gas consumed by the system. In Fabric, this measure is not applicable since there is not such execution cost in a private network. Obviously, this does not mean that the choice of Fabric is for free; indeed, the cost for the hardware required to construct the network should be considered. These costs are highly influenced by the dedicated hardware and the number of desired nodes. Fig. 13 shows the gas consumed for the deployment and execution of the retail process contract in the Ethereum blockchain. The total units of gas used are 6.134.344,00 and it is clear that the deploy transaction is the most expensive one, being around 80% of the total. The remaining transactions are not very impacting and range from a minimum of 77.310 to a maximum of 120.935,00. Of course, the more transactions a contract has, the more gas is consumed. Anyway, on average we have that a general transaction in the Multi-Chain tool, excluding the deployment, consumes around 98.673 units of gas.

7.2. Limitations of the approach

Currently, the main limitations of the approach are given by the blockchain cost to afford during the execution and by the flexibility of the technology to

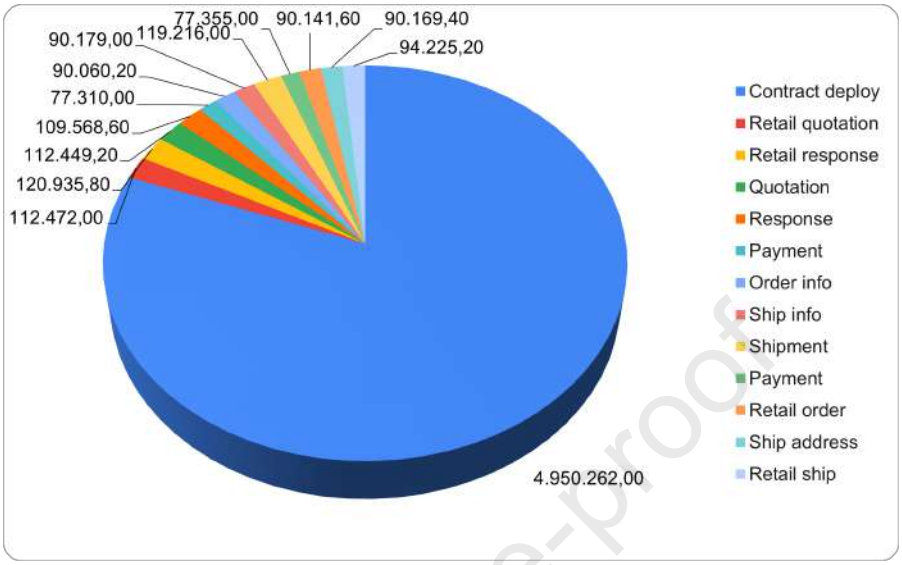

Figure 13: Average gas consumption of a contract.

deal with exceptions and unexpected events. Indeed, for the former point, the use of Ethereum involves a known cost that the user has to pay for each execution/transaction. For what concerns Fabric, there are no fees to pay for the transactions, but it is necessary to host and maintain the network, which surely corresponds to a cost. Moreover, the complexity of the Fabric architecture could create obstacles for the network creation. The latter point is strictly connected to the choice of using the blockchain technology, which does not make it possible to update or change a running instance without a new deployment, thus entailing the loss of the previous interactions.

8. Related works

In literature the blockchains have been used in many contexts and application domains [9]. In this section, we start focusing mainly on reviewing literature regarding model-driven approaches based on choreography-based specifications. Then we focus on those works combining BPMN and the blockchain technology with a focus on business process execution. Finally, we discuss the contributions focusing on multiple blockchains.

The usage of choreography-based specifications to drive the development of multi-party distributed systems has been extensively studied and investigated [3]. Also the EU commission financed various projects specifically de-

voted to the topic (see, for instance, CHOReOS[5] and CHOREVOLUTION[6]). Differently from this research strand, we focus on a specific technology, the blockchain, for supporting the execution of the choreography specifications.

The combination between BPMN and blockchain has been fostered by other works before. In [22], the authors discuss the importance of a model-driven approach to develop a smart contract for blockchain-oriented software. BPMN has been recognised as the most suitable notation for describing smart contracts' behaviour at a higher level of abstraction since it provides facilities for specifying details that help developers and engineers implement the contract interactions. In our work, we use BPMN as well. However, our aim is not to use BPMN to ease smart contracts development, but instead to describe multi-party business processes that will be then implemented in terms of smart contracts. So, despite the fact that the ingredients are the same, the aim of our proposal is quite different.

Other works in the literature recognised blockchain as beneficial for collaborative processes. In [18], the authors outline the potential of blockchain technology to enable a shift in BPM research. They state that large parts of the control flow and business logic of inter-organisational business processes can be compiled from process models into smart contracts, ensuring that the joint process is correctly executed. They summarise technological challenges to address, also providing a smart contract code snippet illustrating how it is generated from a BPMN model, explaining that all state-changing messages have to be recorded in the blockchain and can be accepted only if they are sent from the account registered for the respective role in the process. Despite the novelty of the topic, some concrete implementations of the approach envisioned above can be found in the literature. For example, in [28, 24] BPMN collaboration diagrams are used to provide a framework permitting the execution of decentralised processes exploiting blockchain-related technologies. As in our case, the lack of trust is the main driver for this work. However, the usage of a collaboration model, with the need to provide details for each participating process, constitutes one of the main differences with our proposal. Indeed, we consider choreography diagrams, which are more suitable in a multi parties context, where the internal details of a single organisation are generally not made available. Apart from a different kind of

[5]https://cordis.europa.eu/project/rcn/96288/factsheet/en
[6]http://www.chorevolution.eu/bin/view/Main/

model used to represent the processes cooperation, the approach in [24] introduces a generic factory smart contract that will be reused for each process execution. This introduces a centralisation point, resulting as a bottleneck mainly for reliability issues. Differently, we generate a new contract for each choreography instance, resulting in a clearer separation of concern and simpler management of the information related to the execution of choreography instances. At the same time, the generation of a new contract is distributed on the whole blockchain infrastructure, reducing issues related to scalability and reliability.

Along a similar direction, in [13, 16] the Caterpillar tool is proposed. This is one of the first attempts to support the combination of business process management and blockchain technology. The tool takes as an input a process model and transforms it into Solidity code. Again, the use of a different kind of diagrams distinguishes this proposal from the one we illustrate in this paper, and the same considerations reported above apply here. Extensions of this tool are presented in [14], where the authors propose a dynamic role binding model and a binding policy language for supporting the collaborative business process. In [15], instead, a list of components are provided for the update of models and their smart contracts at runtime, to react to unexpected situations during the execution. With a similar structure, Lorikeet tool is presented in [25], which focuses more on the asset management and business process interactions on the blockchain technology.

The works mentioned above use BPMN collaboration or process models. In fact, BPMN choreographies are still less applied for the other kinds of BMPN diagrams in executing blockchain based scenarios. However, recently, this kind of model has aroused research interest. In [4], the authors present a model-driven approach based on BPMN choreographies, whose target platform relies on a public permissioned blockchain but without a concrete implementation. However, this kind of blockchain still lacks some fundamental properties. Indeed, it is a union of transparency and access control that could be very useful in certain situations but does not fully cover the confidentiality of information and communication. In our case, instead, we have implemented the proposed model-driven approach by giving two possible solutions. Indeed, we support both the public permissionless Ethereum blockchain and the private permissioned Hyperledger Fabric one by taking advantage of these blockchains' characteristics. This clear separation also avoids the user to be insecure about the context in which s/he is operating. Another use is reported in [11], where an extension of the BPMN

choreography is proposed to give more expressiveness to blockchain concepts and implemented in an Ethereum based proof-of-concept. The proposed elements are related to data objects, sub and call choreographies, condition expressions and script tasks. Our approach is quite different, as our work's main goal is to use already existing notation elements to support the full life-cycle in the blockchain, without adding extension elements to the language. The authors highlight also the need of privacy and confidentiality that some business cases could require, pointing to Hyperledger Fabric as a possible solution.

Concerning the works previously described we note that they generally overlook integration aspects related to the need of an infrastructure to support the whole life-cycle of multi parties business processes. Our work, instead, permits to derive a concrete implementation of choreography models, by relying on the underlying blockchain technologies. Our methodology is encapsulated in a user-friendly framework that allows the developer to deal with all the choreography life-cycle phases, from the modelling to the deployment and execution. A web-based interface, easily accessible support all these phases to users not familiar with blockchain-related technologies.

Finally, the works mentioned above target only one type of blockchain, in most cases, Ethereum. Instead, the distinctive characteristics of our work are the capability of supporting multiple blockchains. In practice, our model-driven framework is currently able to automatically generate the code for a given multi-party business process, and deploy and execute it, in both Ethereum and Hyperledger Fabric. The need to consider multiple blockchain technologies has also been exploited in [12], where the authors use choreography diagrams for coordinating the communication between different blockchain technologies. To make it possible, an architecture abstracting from a specific blockchain is proposed. In this way, the communication does not rely on a single technology, but it can integrate heterogeneous technologies, which allows a cross-chain communication. This work aims to generate a bridge between different blockchains. Differently, we propose a methodology and a practical framework for supporting distributed system scenarios on different blockchains, selected based on the system requirements without requesting their integration.

9. Conclusions and Future Work

This work proposes Multi-Chain, a model-driven methodology for multi-party business process on multiple blockchains. It makes possible the automatic generation of distributed systems over blockchain-based technology. In particular, our methodology's starting point is the BPMN standard, which makes it possible to model multi-party business processes from a high-level perspective. Once created, the model system is executed over the blockchain, taking advantage of distribution, trust and immutability of data. This work's principal novelty concerns a flexible framework supporting use cases from different contexts, thanks to the use of both permissionless and permissioned blockchains. In practice, a single blockchain can not suit every need; for example, a transparent public blockchain will not provide privacy and confidentiality. For this reason, the multiple blockchains approach using both Ethereum and the Hyperledger Fabric blockchains, with their complementary properties, allows the users to have complete coverage of his needs. In the proposed tool, we have implemented a dynamic generator and deployer of both networks. In Hyperledger Fabric's case, the framework automatically constructs the appropriate network according to the organisations' specifications.

The paper poses the basis for fast development of multi-party business processes over blockchain technologies, without requiring much technical competence to the final users about blockchain-related aspects. The approach currently includes the most known blockchain technologies between the permissioned and permissionless one. With this work, we highlight the necessity to have different implementations according to the context and the feasibility of the approach that has the ability to deal with rather different technologies. As future work, we intend to extend Multi-Chain to support an automatic selection of the blockchain platform depending on the non-functional requirements of the input choreography and its context of use. We also plan to extend Multi-Chain to cover additional blockchain platforms.

References

[1] Aitor Aldazabal, Terry Baily, Felix Nanclares, Andrey Sadovykh, Christian Hein, and Tom Ritter. Automated model driven development processes. In *Proceedings of the ECMDA workshop on Model Driven Tool and Process Integration*, pages 361–375, 2008.

[2] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, et al. Hyperledger fabric: a distributed operating system for permissioned blockchains. In *Proceedings of the thirteenth EuroSys conference*, pages 1–15. ACM, 2018.

[3] Marco Autili, Paola Inverardi, and Massimo Tivoli. Choreos: large scale choreographies for the future internet. In *Software Evolution Week-IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering (CSMR-WCRE)*, pages 391–394. IEEE, 2014.

[4] Marco Autili, Francesco Gallo, Paola Inverardi, Claudio Pompilio, and Massimo Tivoli. Introducing trust in service-oriented distributed systems through blockchain. In *International Workshop on Governing Adaptive and Unplanned Systems of Systems*, pages 149–154, 2019.

[5] Barbara Carminati, Elena Ferrari, and Christian Rondanini. Blockchain as a platform for secure inter-organizational business processes. In *Collaboration and Internet Computing*, pages 122–129. IEEE, 2018.

[6] Flavio Corradini, Fausto Marcantoni, Andrea Morichetta, Andrea Polini, Barbara Re, and Massimiliano Sampaolo. Enabling auditing of smart contracts through process mining. In *From Software Engineering to Formal Methods and Tools, and Back*, volume 11865 of *LNCS*, pages 467–480. Springer, 2019.

[7] Flavio Corradini, Alessandro Marcelletti, Andrea Morichetta, Andrea Polini, Barbara Re, and Francesco Tiezzi. Engineering trustable choreography-based systems using blockchain. In *35th ACM/SIGAPP Symposium on Applied Computing*, pages 1470–1479. ACM, 2020.

[8] Chris Dannen. *Introducing Ethereum and solidity*, volume 1. Springer, 2017.

[9] Damiano Di Francesco Maesa and Paolo Mori. Blockchain 3.0 applications survey. *Journal of Parallel and Distributed Computing*, 138:99 – 114, 2020.

[10] Archana Prashanth Joshi, Meng Han, and Yan Wang. A survey on security and privacy issues of blockchain technology. *Mathematical foundations of computing*, 1:121, 2018.

[11] Jan Ladleif, Mathias Weske, and Ingo Weber. Modeling and enforcing blockchain-based choreographies. In *International Conference on Business Process Management*, volume 11675 of *LNCS*, pages 69–85. Springer, 2019.

[12] Jan Ladleif, Christian Friedow, and Mathias Weske. An architecture for multi-chain business process choreographies. In *International Conference on Business Information Systems*, volume 389 of *LNCS*, pages 184–196. Springer, 2020.

[13] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, and Ingo Weber. Caterpillar: A blockchain-based business process management system. In *BPM (Demos)*, volume 1920 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2017.

[14] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Dynamic role binding in blockchain-based collaborative business processes. In *Advanced Information Systems Engineering*, volume 11483 of *LNCS*, pages 399–414. Springer, 2019.

[15] Orlenys López-Pintado, Marlon Dumas, Luciano García-Bañuelos, and Ingo Weber. Interpreted execution of business process models on blockchain. In *23rd International Enterprise Distributed Object Computing Conference*, pages 206–215. IEEE, 2019.

[16] Orlenys López-Pintado, Luciano García-Bañuelos, Marlon Dumas, Ingo Weber, and Alexander Ponomarev. Caterpillar: a business process execution engine on the ethereum blockchain. *Software: Practice and Experience*, 49(7):1162–1193, 2019.

[17] Mads Frederik Madsen, Mikkel Gaub, Tróndur Høgnason, Malthe Ettrup Kirkbro, Tijs Slaats, and Søren Debois. Collaboration among adversaries: distributed workflow execution on a blockchain. In *Symposium on Foundations and Applications of Blockchain*, pages 1–8, 2018.

[18] Jan Mendling, Ingo Weber, Wil Van Der Aalst, Jan Vom Brocke, Cristina Cabanillas, Florian Daniel, Søren Debois, Claudio Di Ciccio, Marlon Dumas, Schahram Dustdar, et al. Blockchains for business process management-challenges and opportunities. *ACM Transactions on Management Information Systems (TMIS)*, 9(1):1–16, 2018.

[19] Business Process Model OMG. BPMN, 2011.

[20] Oscar Pastor. Model-driven development in practice: From requirements to code. In *International Conference on Current Trends in Theory and Practice of Informatics*, volume 10139 of *LNCS*, pages 405–410. Springer, 2017.

[21] Julien Polge, Jérémy Robert, and Yves Le Traon. Permissioned blockchain frameworks in the industry: A comparison. *ICT Express*, 2020.

[22] Henrique Rocha and Stéphane Ducasse. Preliminary steps towards modeling blockchain oriented software. In *1st International Workshop on Emerging Trends in Software Engineering for Blockchain*, pages 52–57. IEEE, 2018.

[23] P Sajana, M Sindhu, and M Sethumadhavan. On blockchain applications: Hyperledger fabric and ethereum. *International Journal of Pure and Applied Mathematics*, 118(18):2965–2970, 2018.

[24] Christian Sturm, Jonas Szalanczi, Stefan Schönig, and Stefan Jablonski. A lean architecture for blockchain based decentralized process execution. In *Business Process Management Workshops*, volume 342 of *LNBIP*, pages 361–373. Springer, 2018.

[25] An Binh Tran, Qinghua Lu, and Ingo Weber. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In *BPM Dissertation Award, Demonstration, and Industrial Track*, volume 2196, pages 56–60. CEUR-WS.org, 2018.

[26] Wattana Viriyasitavat and Danupol Hoonsopon. Blockchain characteristics and consensus in modern business processes. *Journal of Industrial Information Integration*, 13:32–39, 2019.

[27] Marko Vukolić. Hyperledger fabric: towards scalable blockchain for business. *Trust in Digital Life*, 2016.

[28] Ingo Weber, Xiwei Xu, Régis Riveret, Guido Governatori, Alexander Ponomarev, and Jan Mendling. Untrusted business process monitoring and execution using blockchain. In *International Conference on Business*

Process Management, volume 9850 of *LNCS*, pages 329–347. Springer, 2016.

[29] Xiwei Xu, Ingo Weber, Mark Staples, Liming Zhu, Jan Bosch, Len Bass, Cesare Pautasso, and Paul Rimb. A taxonomy of blockchain-based systems for architecture design. In *International Conference on Software Architecture*, pages 243–252. IEEE, 2017.

Declaration of interests

X 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.

☐The authors declare the following financial interests/personal relationships which may be considered as potential competing interests: