Engineering Trustable Choreography-based Systems using Blockchain

F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi
School of Science and Technology, University of Camerino
{name.surname}@unicam.it

ABSTRACT

The adoption of model-driven engineering methodologies contributes to reduce the complexity of developing distributed systems. A key point to master such complexity is the use of modelling languages, such as the BPMN standard. This permits to specify choreography diagrams describing, from a global point of view, the interactions that should occur among distributed components in order to reach given goals. Even though BPMN choreographies are promising to increase business possibilities, their concrete adoption has been challenging and faced complex hurdles. On the one hand, there is a lack of concrete support to the different phases of the choreography life-cycle, especially in relation to the choreography execution. Another obstacle consists in the lack of distributed infrastructures allowing the participants involved in the cooperation to trust each other, and in particular to get enough guarantees that all of them will behave as prescribed by the choreography model.

In this paper, we face such challenges by proposing a methodology and a related model-driven framework, named ChorChain, that are based on the blockchain technology. We provide support to the whole life-cycle of choreographies, from their modelling to their distributed execution. More specifically, ChorChain takes as input a BPMN choreography model and automatically translates it in a Solidity smart contract. Such a contract will permit to enforce the interactions among the cooperating participants, so to satisfy the prescriptions reported in the starting model. The methodology and the framework have been evaluated through experiments conducted on the Rinkeby Ethereum Testnet.

ACM Reference Format:
F. Corradini, A. Marcelletti, A. Morichetta, A. Polini, B. Re, F. Tiezzi . 2020. Engineering Trustable Choreography-based, Systems using Blockchain. In *The 35th ACM/SIGAPP Symposium on Applied Computing (SAC '20), March 30-April 3, 2020, Brno, Czech Republic.* ACM, New York, NY, USA, 10 pages. https://doi.org/10.1145/3341105.3373988

1 INTRODUCTION

The development of advanced distributed systems is made complex by the need of coordinating the interactions of various components, possibly controlled by different organisations. Model-driven engineering methodologies help to master this complexity by means of modelling languages conceived to describe the interaction logic of the overall applications, thus abstracting from technical implementation details. These methodologies then allow to pass from abstract specifications to code executable on a targeted runtime environment.

Nowadays, a prominent modelling language to describe distributed collaborative systems is the BPMN standard [23]. Although it has been initially introduced for documentation of business processes, now the use of BPMN models is advocated as a starting point for the model-driven engineering of distributed systems [1, 24]. BPMN provides a set of flow chart based notations, permitting to represent different perspectives on business processes and their cooperation. In this paper, we focus on *choreography* diagrams that permit to describe system interactions in terms of the exchange of messages from a global perspective, without exposing the internal behaviour of the participants.

Despite the wide acceptance of BPMN from both industry and academia, in practice until now the usage of such diagrams have been mainly confined to the design phase [22]. Indeed there is a lack of a concrete support to the other phases, in particular in relation to *choreography execution*. In addition, the full adoption of choreography specifications for the engineering of distributed systems has been hindered by the difficulty of ensuring *trust*[1] among the distributed participants [30], which indeed will engage in the cooperation only if they trust each other. Alternatively they can possibly resort to a third-party trusted authority that guarantees the effective exchange of the messages prescribed by the model.

In this paper, we aim at addressing the above issues by leveraging on the *blockchain* technology, as envisioned in [22]. Blockchain is an emerging technology for decentralised and transactional data sharing across a network of untrusted participants. It guarantees the integrity and the immutability of data without relying on a central authority or any particular participant [8, 20]. Thus, blockchain enables the development of new forms of distributed systems, where all participants have a clear view of the ongoing system execution and can have a tangible proof of the actions performed by the counterpart. In particular, in our choreography-based setting, the blockchain technology is exploited to allow choreography participants to achieve trust without a central authority, by enabling the auditing of the choreography execution in a non-repudiable manner. In this way, for example, it can be checked that a given participant has actually sent/received a given message as specified in the choreography.

The main challenges [3, 25, 26] in relation to the development of choreography-based distributed systems on top of blockchain

, ,

ACM ISBN 978-1-4503-6866-7/20/03…$15.00
https://doi.org/10.1145/3341105.3373988

[1] We refer here to a general notion of trust in business relationships, as defined in [27] as"the belief that a party's word or promise is reliable and that a party will fulfil his/her obligations in an exchange relationship".

are: *(i)* the need to fill the gap between the high-level description given by the choreography, and the low-level one given by the code deployed and executed on the blockchain, without annoying the end user with convoluted technicalities; *(ii)* to support the whole life-cycle of choreographies, from their design to their execution in the blockchain. We face these challenges by defining **a model-driven methodology and** providing **a practical framework, called *ChorChain*,[2] for the development of trustable choreography-based systems.** ChorChain exploits the blockchain technology, in particular Ethereum [32], as the base for executing BPMN choreographies. The proposed model-driven engineering methodology supports the full life-cycle of choreographies, from their modelling to their publishing and instantiation, until their deployment and execution in the Ethereum blockchain. In particular, ChorChain supports the automatic generation of *smart contracts* (i.e., programs to be run over the blockchain), which is completely transparent to the final user, and the controlled execution of the choreography, which allows only the sequence of actions prescribed by the specification.

The smart contract executed in the blockchain, indeed, is specifically generated to implement the choreography workflow, thus forcing the correct behaviour of each participant. Moreover, the smart contract generated by ChorChain can manage payments in Ether as possibly specified in the choreography model. In addition, the guarantees on data stored in the blockchain allow the auditing of exchanged messages during the choreography execution, without the need of a monitoring infrastructure. On the other hand, the trust established among the choreography participants comes at a cost: the creation of the smart contract in the blockchain where the execution of each choreography action will correspond to a price to be paid to encourage computation support in Ethereum. We discuss the (affordable) cost for executing a smart contract generated by ChorChain in Sect. 5.

To sum up, ChorChain acts as a Business Process Management System for tamper-proof choreographies execution, differentiating from approaches available in the literature with similar objectives (such as, e.g., those introduced in [18, 29]). In particular differences relate to the modelling abstraction adopted (i.e., choreographies, neither processes nor collaborations), the usability of the resulting tool, and to the comprehensive functionalities covering the whole choreography life-cycle.

The rest of the paper is organised as follows. Sect. 2 provides an overview of the BPMN notation and the Ethereum blockchain. Sect. 3 introduces the proposed methodology, while Sect. 4 illustrates the ChorChain framework. Sect. 5 presents the results of the validation experiments carried out on the Rinkeby Ethereum Testnet. Sect. 6 reviews related works. Finally, Sect. 7 concludes by touching upon directions for future work.

2 BACKGROUND NOTIONS

Choreographies. Fig. 1 depicts the most used modelling elements that can be included in a BPMN choreography diagram. On the left, we depict the elements used to define the control flow, while on the right, the elements used to represent the communication tasks. **Events** can be used to represent the start (*start events*) or the end (*end events*) of a choreography. **Gateways** are used to drive the flow of a choreography. They can act either as join nodes (merging incoming sequence edges) or split nodes (forking the flow into multiple outgoing edges). Different types of gateways are available. An *exclusive gateway (XOR)* permits to represent choices; it is activated each time the gateway is reached in join mode and, in split mode, it activates exactly one outgoing edge. A *parallel gateway (AND)* in join mode waits to be reached by all its incoming edges to start; instead, in split mode, all the outgoing edges are started simultaneously. An *event-based gateway* activates its outgoing branches in relation to the reception of the message associated with the different branches. Indeed, the associated message events are in a race condition, and the first one that is triggered will activate the branch, also disabling the other ones. **Sequence Flows** are edges used to connect events, gateways and tasks permitting to specify the choreography execution flow. **Tasks** are used to specify the message exchange between two participants. They are drawn as rectangles divided into three bands: the central one includes the name of the task, while the other two refer to the involved participants (the one in white is the initiator, while the grey one is the recipient). Messages can be sent either by one participant (One-Way tasks) or by both participants (Two-Way tasks).

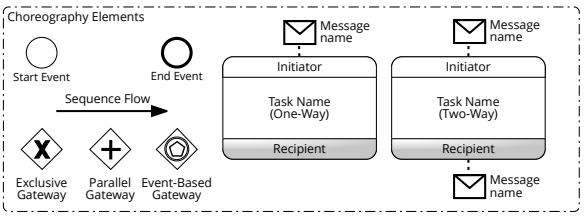

Figure 1: BPMN Choreography Elements.

Blockchain. A blockchain is a distributed ledger composed by a linked list (cf. chain) of records called blocks. Each block contains a limited number of transactions in its body, while the header includes, among the others, the hash of the current block and the hash of the previous block. New blocks are added to the chain at regular interval of time by the so-called "miners". These are computational nodes, related to the blockchain infrastructure, that are needed to derive the hash of a block. The mining process and the use of consensus protocols permit to verify the genuineness of the transactions included in each block. Finally, the replication of the chain in any node of the network guarantees decentralisation and trustworthiness, without the need of a third party independent authority. The blockchain ideas have been initially proposed to support payment systems based on cryptocurrencies. However, in the last years, blockchain technologies have been adopted in many other different contexts, especially those blockchain platforms that support such advanced features as the *smart contracts*. These can be considered as special programs which are executed over the blockchain infrastructure, whose nodes are now equipped, in some specific technologies such as Ethereum, with computational power. The execution of smart contracts produces transactions that are stored in the blockchain, thus making their execution auditable.

Ethereum blockchain. Ethereum is a concrete implementation of the blockchain that includes support for the execution of smart contracts. This is the technology we used to implement the

[2]The ChorChain framework is available at http://pros.unicam.it/chorchain/.

ChorChain framework. In Ethereum, every node connected to the Ethereum network embeds an instance of the Ethereum Virtual Machine (EVM). The operations executed in the EVM, like storage of information or contract instructions, have an associated economic cost defined in terms of *gas*, which is the unit measuring the amount of computational effort for the execution of the operation. The execution cost has two main advantages: it reduces the risk of malicious computational tasks, and it encourages mining activities by network participants and, hence, it permits to keep the overall system working. Indeed, miners are rewarded for each block they mine with a default amount of Ethers plus the sum of the transaction fees included in the block. The most prominent language to write smart contracts for Ethereum is *Solidity* (https://solidity.readthedocs.io/). It is worth mentioning that any user in the Ethereum network can be recognised with its account uniquely identified by a hex address. We exploit this feature in our choreography execution to identify participants.

3 THE CHORCHAIN METHODOLOGY

In this section, we focus on the methodological aspects of the approach we propose, while a technical description of how the various phases of the choreography life-cycle are practically supported by the ChorChain framework (Fig. 2) is given in the next section.

The first phase of our model-driven methodology is the **system modelling** one, consisting in the creation of a choreography specification. The main motivation for using a specification of how the different system components should interact to reach a common goal concerns the possibility to abstract from implementation details. In our blockchain-based solution, in particular, the use of an abstract specification permits to alleviate the burden on the shoulders of the developer, who can avoid to directly deal with smart contract technicalities.

The resulting **model is published in a choreography repository** making it publicly available, and enabling the **searching** phase. The open access to models can foster cooperation among unrelated parties. In our approach, a choreography model is conceived as a blueprint, which can be **instantiated** to activate multiple cooperations with the same structure, but possibly involving different participants. To increase the reusability of the choreography models, the instantiator can indicate which roles in the newly created instance are mandatory and which are optional. This form of flexibility permits to use the same choreography in slightly different scenarios; for example, a choreography supporting two different ways of payment, i.e. bank transfers and PayPal, can be instantiated by requiring both roles as mandatory or by indicating one of them as optional. It is a duty of the initiator to accurately select the mandatory/optional roles in order to ensure the proper completion of the instance execution. When a choreography instance is created, in order to be executed it has to be **subscribed by** the participants that aim at playing a given role in that specific cooperation. Before passing to the next phase, it is necessary that each mandatory role has been subscribed by a participant.

Once all the required roles are filled, the smart contract corresponding to the choreography instance is **generated** and then **deployed**, automatically, on the blockchain. At that time, the **execution** phase starts, and the participants can cooperate following the message protocol established by the choreography specification, and implemented in the smart contract. The smart contract permits to ensure that the participant interactions are compliant with the choreography specification, since only the enabled actions are allowed to the partner in charge for their execution. Moreover, each performed action results in a transaction stored in the chain, enabling successive **auditing** activities.

4 THE CHORCHAIN FRAMEWORK

In this section, we present the ChorChain framework, by focussing on the design choices that have driven its development, and the technical solutions we have adopted for its implementation. We organise the presentation according to the phases of the methodology introduced in Sect. 3. The reader can practically experiment with the ChorChain framework deployed at http://virtualpros.unicam.it:8080/ChorChain using the Rinkeby Ethereum Testnet (https://www.rinkeby.io/), which is a sandbox copy of the Ethereum blockchain.

Modelling. The modelling phase is the starting point of the choreography life-cycle. To support it we have chosen to integrate a modelling environment into the framework. This choice permits to avoid the notorious interoperability issues of BPMN modelling environments, thus guaranteeing full compatibility of the produced choreography model with all the features provided by the rest of the framework. The modelling area offers several functionalities, starting from the creation and update of a choreography model, the export of the model as an image or a .bpmn file, the import of an existing model, and finally the storage of the choreography on the ChorChain repository.

However, a choreography model, due to its level of abstraction, does not contain all the information needed for its execution. To overcome such limitation, ChorChain asks to the modeller additional data related to *(i)* messages and *(ii)* guards. Thus, during the modelling phase the modeller has to annotate the message(s) of each choreography task with the parameters needed for performing the underlying function call in the generated smart contract. To facilitate this procedure, in the integrated modelling environment 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 message parameters, and the indication if the message contains a payment. The result of this procedure is the addition to the model of a list of parameters after the message name; for example, the annotation *request*(*uint orderId*, *uint quantity*, *string item*) defines a message named *request* consisting of two unsigned integers (i.e., the order identifier and the quantity of the ordered item) and a string (i.e., the name of the item).

Another fundamental aspect to consider when executing a choreography is related to the guards of its exclusive gateways. Indeed, each sequence flow outgoing from an exclusive gateway must refer to a boolean expression involving variables already defined in the model. This is necessary to guarantee a proper execution of the choreography model when deployed in the blockchain.

Publishing, Searching and Instantiation. To publish the choreography model, ChorChain provides a repository that can be accessed via an intuitive user interface. However, in order to

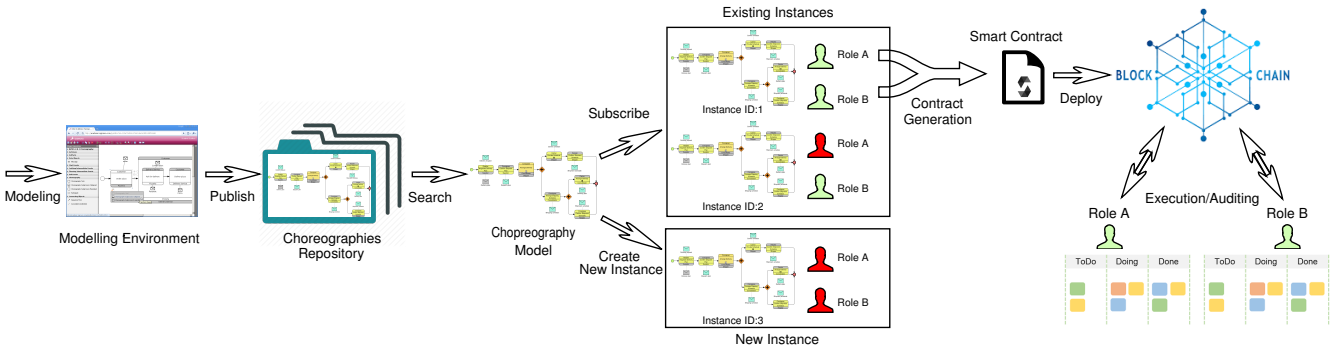

Figure 2: Choreography life-cycle supported by ChorChain.

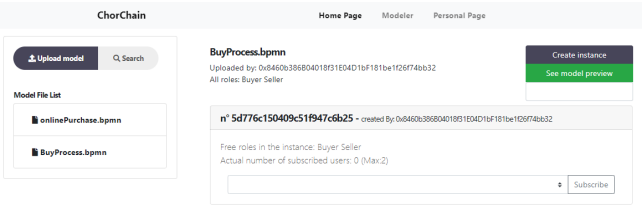

Figure 3: ChorChain Home Page.

interact with the repository it is necessary to register and login into the platform. To ease the access to the platform, the user can exploit the Metamask browser plugin (https://metamask.io/), which provides a web interface for managing Ethereum accounts. The account selected in Metamask will then constitute the identifier of the participant in a choreography execution. After the login, the user is redirected to the homepage depicted in Fig. 3. On the left side of the web page, the user has the possibility to publish a new model, by uploading the corresponding file, or to search for an existing one.

The searching phase is an important aspect of the framework, since it enables reusability and facilitates the meeting between supply and demand. Any registered user, once logged in, can search for a particular choreography name and the framework proposes the list of all models matching the searched topic. These models are listed below the search form (bottom left corner in Fig. 3). At this stage, we only provide a simple discovery functionality, as this is not the main focus of our work. More advanced mechanisms can be adopted, by resorting, for instance, to semantic annotations of choreography models.

The information about the selected choreography model is shown on the right side of the homepage. In particular, ChorChain shows: the owner of the model, the maximum number of involved participants and the required roles. The preview of the graphical representation of the model, and the possibility to create a new instance, is also visualised.

In the instantiation phase, it is possible to select one or more roles and mark them as optional, otherwise they are considered mandatory. Consequently a choreography instance will be kept in a "suspended" state while waiting that all the mandatory roles are subscribed. To fill these roles, the instance creator can send the link of the "suspended" instance to the participants he intends to involve, or to other possible participants differently identified. It is possible to support public instances, where participants are free to enter, and private instances where only preselected parties have the opportunity to join. Fig. 2 shows three instances of the same choreography composed of two mandatory roles (Role A and Role B). Instance 1 is the one deployable on the blockchain, since both roles are subscribed (green colour). Instances 2 and 3 are suspended: the former is incomplete, since just one role is subscribed (Role B) and one is vacant (Role A, in red colour), while the latter instance is newly created and all its roles are vacant.

When an instance has no more vacant mandatory roles, ChorChain considers the partnership complete and starts the generation and deployment of the Solidity smart contract.

Smart Contract Generation. The smart contract generation is an automatic phase where the choreography instance is translated into Solidity code. The generation of the contract structure starts after the parsing of the choreography model, performed by means of the Camunda library (https://docs.camunda.org/javadoc/camunda-bpm-platform/7.11/), extended by us to deal with choreography diagrams. Here below we give some hints of the automatic translation performed by ChorChain for each choreography element admitted by the BPMN standard.

Listing 1 reports the header of the contract. This part is quite standard and similar for each newly generated instance.

```
1  contract Choreography{
2
3    enum State {DISABLED, ENABLED, DONE} State s;
4    struct Element{string ID; State status;}
5    struct StateMemory{uint t,z;}
6    event stateChanged(uint eventID);
7
8    Element[]   choreographyElements;
9    StateMemory currentMemory;
10
11   mapping (string=>uint) position;
12   string[] elementsID=["elementID", ..."elementID"];
13
14   mapping(string=>address) roles;
15   mapping(string=>address) optionalRoles;
16   string[] roleList = ["x",...,"y"];
17   string[] optionalList = ["t",...,"z"];
```

Listing 1: Contract Header.

Here the contract keeps track of the choreography state by means of the list of elements `choreographyElements` (line 8) and the structure of variables `currentMemory` (line 9) 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) containing instead 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 `stateChanged` (line 6) is used to notify the partners about the state change of the contract. In the header we also have the list of identifiers of all elements in the choreography model `elementsID` (line 12), and the list of mandatory and optional roles involved in the choreography `roleList` (lines 16-17).

After the declaration of the global variables, in the contract we have the definition of the modifiers (Listing 2).

```
16 modifier checkMand(string memory role) {
17   require(msg.sender == roles[role]);
18   _;
19 }
20 modifier checkOpt(string memory role) {
21   require(msg.sender == optionalRoles[role]);
22   _;
23 }
```

Listing 2: Modifiers.

A modifier has the same structure of a method, but can be called only inside the definition of a function. Specifically, the modifiers `checkMand` (lines 16-19) and `checkOpt`(lines 20-23) 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.

The contract constructor (Listing 3) is the principal method executed at the contract deployment time. It performs all the operations concerning variables initialization, necessary for the subsequent execution of the contract.

```
20 constructor() public{
21   for (uint i = 0; i < elementsID.length; i++) {
22     choreographyElements.push(Element
23           (elementsID[i], State.DISABLED));
24     position[elementsID[i]]=i;
25   }
26   //roles definition
27   roles["Buyer"] = 0x0000000...;
28     ...
29   optionalRoles["Seller"] = ...;
30   init();
31 }
32
33 function sub_as_part(string memory _role) public {
34   if(optionalRoles[_role]==0x0000000...){
35     optionalRoles[_role]=msg.sender;
36   }
37 }
```

Listing 3: Contract Constructor.

The first operation performed by the constructor is the initialization of the `choreographyElements` list with the default value `DISABLED` (lines 21-25). Lines 27-29 report the initialisation of the roles with the addresses of the participants. The mandatory roles are hard-coded in the constructor, since they are roles without which the execution of the choreography model cannot take place. For the optional roles, instead, the participants can subscribe at run-time using the `sub_as_part` function in (lines 33-36).

After the generation of this first part of the contract, which is similar for each choreography model, the generation continues by appending the definitions of all elements effectively included in the considered choreography model. In the contract generation, 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 in two main categories: *messages*, representing the interactions between participants, and *control flow* elements. The translation generates a public function for each task message, and a private function for each element of the control flow of the choreography.

Listing 4 shows a public function depicting a message `offer` of a choreography task. The function name is inherited from the message identifier, while the parameters from the annotated name.

```
39 function offer(uint price) public
40                checkMand(roleList[1]) {
41   require(elements[position["currentElement"]].status
42           ==State.ENABLED);
43   currentMemory.new_price=price;
44   ...
45   done("currentElement");
46   enable("nextElement");
47 }
```

Listing 4: A Message Function.

The modifier `checkMand` is called with the assigned role (line 40). Once the right identity of the caller inside the function is ascertained, a second check on the status of the task is performed: as expected it should be equal to `ENABLED` (lines 41-42). After that, it is performed the registration of the `price` parameter in the memory of the contract (line 43). The last step before enabling the execution of the next element is to change the status of the current element to `DONE` (line 45) and set to `ENABLED` the status of the next element (line 46).

In Listing 5 there is the representation of a *parallel gateway* element.

```
48 function parallelGateway() private {
49   require(elements[position["currentElement"]].status
50           ==State.ENABLED);
51   done("currentElement");
52   enable("nextElement_1");
53   ...
54   enable("nextElement_n");
55 }
```

Listing 5: Function for a Parallel Gateway.

This function is private, hence it cannot be directly called by the choreography partners, but only from inside the contract. This guarantees that the control flow of the model is only managed by the contract and not influenced by external users. The other characteristic of the control flow functions is the absence of the modifier for the role checking, since there is no interaction coming from outside the blockchain. The function has no parameters, since the execution semantics of the corresponding element does not require external inputs. The body of the function contains just the code to complete the current element (line 51) and that to enable the next elements connected to the parallel gateway (lines 52-54). A similar

code can be expected for an *event-based gateway*, as in principle all the tasks connected to that gateway has to be enabled and only when the message is received the task waiting for the antagonistic messages has to be disabled. A different implementation is expected for an *exclusive gateway*, where the logic of the function is depicted in Listing 6. In this case the next element is enabled only after the evaluation of a condition that discriminates which element to enable.

```
56  function exclusiveGateway() private {
57    require(elements[position["currentElement"]].status
58            ==State.ENABLED);
59    if(condition){
60      enable("nextElement_1");
61    }else{
62      enable("nextElement_2");
63    }
64    done("currentElement");
65  }
```

Listing 6: Function for an Exclusive Gateway.

State changes for the elements is simplified by the usage of three auxiliary functions defined in Listing 7. These functions are similar: each one takes as input an element identifier and, after a search inside the `choreographyElements` list, it changes the status of the element to `ENABLED`, `DISABLED` or `DONE`, respectively. The `enable` function is the only one with event emission (line 69). In particular, each time a new element is enabled an event is triggered for notifying the participants about the new state in the contract.

```
66  function enable(string memory _ID) internal {
67    choreographyElements[position[_ID]].status
68    =State.ENABLED;
69    emit stateChanged(eventID);
70  }
71
72  function disable(string memory _ID) internal {
73    choreographyElements[position[_ID]].status
74    =State.DISABLED;
75  }
76
77  function done(string memory _ID) internal {
78    choreographyElements[position[_ID]].status
79    =State.DONE;
80  }
```

Listing 7: State Change Functions.

Finally, a specific function, reported in Listing 8, is used for dealing with payments.

```
48  function payment(address payable to) public payable checkMand
        (roleList[0]){
49      require(elements[position["currentElement"]].status==
            State.ENABLED);
50      to.transfer(msg.value);
51      done("currentElement");
52      enable("nextElement");
53  }
```

Listing 8: Function for a payment.

This function is used to send money to a particular address passed as a parameter. The function is marked as `payable`, and allows to transfer Ether from the account of the participant executing it to the passed account. The `transfer` function (line 50) sends to the addressee an amount of Ether specified by `msg.value`, which is defined by the sender user.

Deployment. Once the contract has been generated, the framework automatically deploys it into the Ethereum blockchain. To get a better understanding of this process, we first clarify the architectural organisation of ChorChain. The framework has the typical *DApp* architecture (Fig. 4), including the following components: front-end, back-end and database. The user interacts with the framework via the *front-end*, which is a web application developed in AngularJS. The *back-end* instead takes care of the management of choreography models and instances, and of the generation, deployment and execution of smart contracts. These functionalities are mainly implemented in Java using the Spring Framework. During the choreography execution a role can be played by two different types of participants: human users, acting via the front-end interface, or software components, acting via the REST API provided by the back-end. Finally, the *database* component is used for storing choreography models and their instantiations. This component is realised using the mongoDB database and the Hibernate framework.

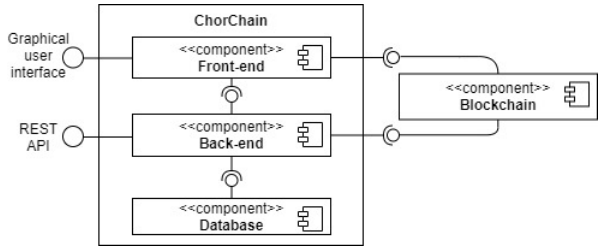

Figure 4: ChorChain Architecture.

The interactions between a participant, the ChorChain components and the Ethereum blockchain, for the subscription to an instance and the deployment of the corresponding contract, can be now summarised as depicted in Fig. 5. Given a suspended instance, the participant can subscribe to a role by sending a request to the back-end, through the front-end. At this point, two alternatives are admitted. In case all the mandatory roles are covered, the ChorChain back-end generates the smart contract, by sending a transaction to the blockchain using the *web3j* library (https://web3j.io). The Ethereum blockchain uniquely identifies the contract with a hexadecimal number, which is generated only after the corresponding transaction is mined. The contract creation event is then caught by the back-end, which will update the front-end, and it will share the contract identifier among all participants. The choreography execution can now be started. The other alternative in the sequence flow describes the case where the instance is still incomplete. In this case, the back-end just updates the subscribed role without involving the blockchain.

Execution. Once a new contract is deployed into the blockchain, the execution phase takes place and the participants can collaborate by means of the functions exposed by the contract. In order to facilitate these interactions, ChorChain provides each (human) participant with a personal area (Fig. 7). The left hand side of this interface reports a list of all contracts, to which the participant is subscribed. In the right hand side, it is shown a preview of the model with the active messages in green colour, with below the respective form that is dynamically constructed by ChorChain for executing it. The form contains several information like: the name of the message, the role of the participant, the space for inserting all the required parameters, plus the submit button. Notably, the

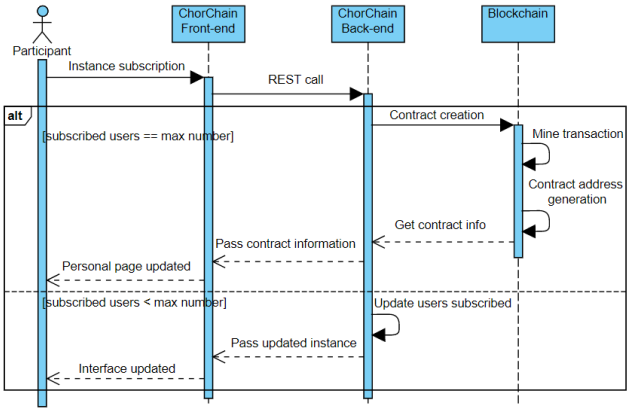

Figure 5: ChorChain Deployment phase.

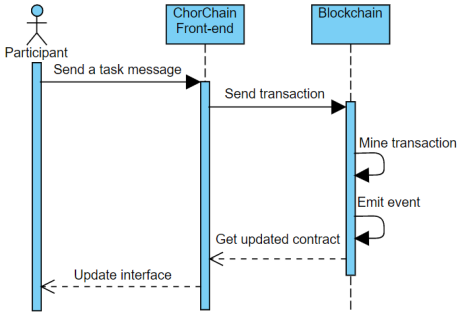

Figure 6: ChorChain Execution Phase.

submission form is visible only to the participant in charge of sending the enabled message. A table, in the bottom-right corner, shows the current value of the variables involved in the contract.

The sequence diagram reported in Fig. 6 summarises the steps for the execution of a (one-way) task. To send the task message, the participant has to fill-up the corresponding form, and then click on the submit button. The generated transaction has to be confirmed using an automatically triggered 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. This event is used by the front-end to update the interfaces of all participants involved in the choreography with the new contract status, thus enabling the next admitted message(s). 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 the back-end component.

5 VALIDATION

To illustrate how our approach works in practice and to assess the effectiveness of the ChorChain framework, we performed some experiments based on the Online Purchase scenario depicted in Fig. 8 (available at http://bit.ly/ChorChainRep). We do not provide a detailed description of the model, as the meaning of each task should be intuitive, while for that of the control flow elements we redirect the reader to the description in Sect. 2.

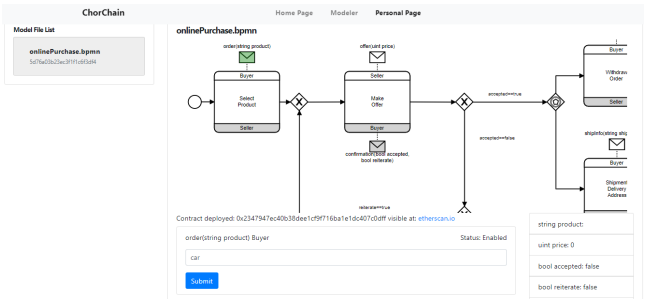

Figure 7: ChorChain Participant Personal Area.

Following the proposed methodology, the choreography model was published into the ChorChain framework, then a new choreography instance was generated and, after the subscription of participants, the Solidity contract was deployed. The choreography contains two participants, and each of them was subscribed with a different account. The experiments had been performed on September 10th on the Rinkeby Testnet blockchain. The reader can find the complete version of the generated Solidity contract at the link http://bit.ly/ChorChainRep and the executed transactions publicly accessible on the Etherscan website at the link http://bit.ly/ChorChainEth. Looking at the transactions, we find first the contract creation, and then the transactions related to the control flow and the exchange of messages between participants.

In Fig. 9 we report the cost analysis for the Online Purchase contract execution. For each transaction, we list its name, the used gas, and the fee (both in Ether and in US dollars). Notably, we report the execution related to the longest execution path. As expected, the first recorded transaction is related to the contract creation. It is the most expensive transaction, which costs around 0.006 Ether corresponding to $ 1.11 at the current exchange rate of $ 178.51. All the other transactions represent the exchange of messages between participants. They are rather cheap, since the used gas ranges from a minimum of 48126 to a maximum of 159920. Since the gas price for the transaction was set to 1 Gwei, the execution of the Online Purchase choreography in Ethereum using ChorChain consumes at maximum around 7.1 millions of gas and around $ 1.2 of fee. The user can improve the time for the transactions registration by increasing the gas price. However, the obtained benefits are limited by technology constrains given by the Rinkeby Testnet. To have a better transaction inclusion throughput the ChorChain framework could be deployed in a private Ethereum blockchain. Nevertheless, we consider the cost of using the public net still reasonable in return of trust. Moreover, we observe that the cost is mainly influenced by the contract creation, while the number of exchanged messages does not have a relevant impact.

To provide a more detailed evaluation of the proposed approach in terms of gas consumption, we show here the results of two additional experiments. The first one is still based on the Online Purchase example and is used for testing the total cost of the admissible paths in the choreography, while the second is more general and aims at clarifying the scalability of the framework considering the impact of the number of messages in terms of gas during the contract creation (which is indeed the most expensive phase). The

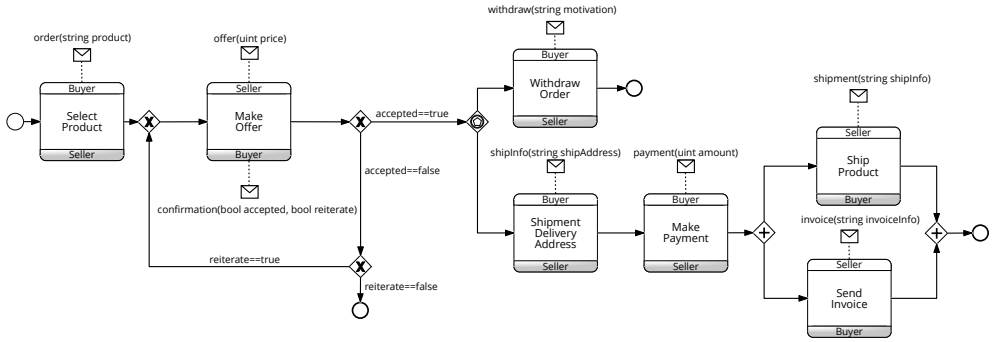

Figure 8: Online Purchase Choreography.

Transaction Name	Gas Used	Transaction Fee in Ether	Transaction Fee in US$
contract creation	6208580	0.00620858	1.11
order	114353	0.000114353	0,020
offer	78126	0.000078126	0.014
confirmation	159920	0.00015992	0.029
offer	48126	0.000048126	0.0086
confirmation	152015	0.000152015	0.027
shipInfo	71120	0.00007112	0.013
payment	129335	0.000129335	0.023
invoice	88194	0.000088194	0.016
shipment	108780	0.00010878	0.019

Figure 9: Online Purchase Cost Analysis.

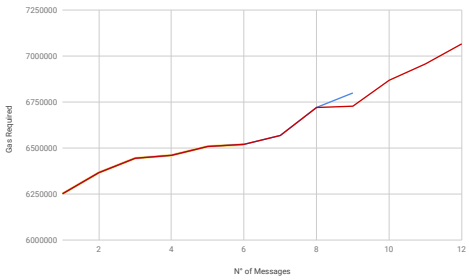

Figure 10: Cumulative gas consumption in the Online Purchase choreography.

chart in Fig. 10 depicts the cumulative gas used for the execution of the Online Purchase choreography, considering the three possible completions admitted in the model and including one iteration in the Make Order loop. The yellow line represents the cost of the path for reaching the end event after the exclusive gateway in the middle bottom of Fig. 8, the line blue refers to the path leading to the withdraw of the order, and the red one refers to the path leading to the successful completion. Summing up, the amount of gas used for executing a complete path in the choreography is reasonable in terms of transactions, although it is required a high cost for the contract creation. For this reason, in Fig. 11 we report a scalability analysis focused on the impact of the contract creation concerning the number of messages contained in the model. The graph shows that the used gas increases linearly with the number of messages. This trend is encouraging, since it can be kept under control. However, we should consider that the current Ethereum block gas limit in the Rinkeby network is around 10 millions, which hence sets an upper bound to the number of messages that can

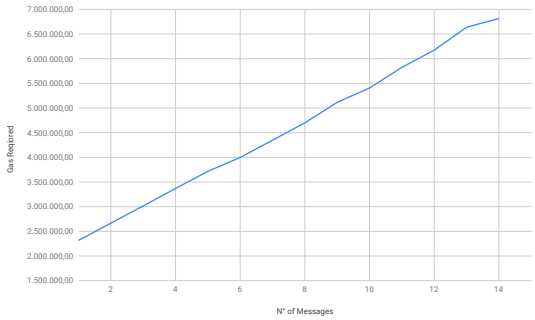

Figure 11: Scalability experiment.

be exchanged for each contract. This limitation could be simply overcome by splitting the contract into separate sub-contracts.

6 RELATED WORKS

The usage of choreography-based specifications, to drive the development of multi-party distributed systems, has been extensively studied and investigated. In particular, in the last years the EU commission financed various projects specifically devoted to the topic (see, for instance, CHOReOS[3] [6] and CHOREVOLUTION[4]). The relevance of choreographies in relation to the description of complex inter-organisational systems, and the need of suitable infrastructures for the management of the choreography life-cycle, is discussed in [2]. In particular the authors propose ServicePot, a complex choreography registry elevating choreography specifications to first class citizens, in order to facilitate the dynamic integration and interoperability of services managed and made available by different organisations. A correlated work is the one in [4], where the authors propose an automatic approach for enforcing the realisability of choreographies providing adapters. Starting from a choreography model and a set of services, the proposed adaptation and coordination solution allows services to collaborate according to the choreography specification. In [7] the authors propose a correct-by-construction method to build realisable choreographies described using conversation protocols, while in [10] the authors propose an approach for checking conformance of a possible system implementations, with respect to choreographies.

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

Similarly, the use of the blockchain technology in the development of decentralised applications is discussed thoroughly in the literature [31] and different works moves in this direction, reinforcing the motivations that led our effort in this context. In particular, in [13] the authors propose an application for automatic gasoline purchase to highlight the concrete possibilities and limitations related to the adoption of blockchain technologies. Among the various possibilities, the authors identify three main important aspects that can be synthesised in: transparency, longevity and trust. The latter is a key aspect also in our work. Other concrete applications can be found in environments where the innate characteristics of this new technology can replace old system components, like in [33] where the authors propose a design approach for building a blockchain-based product traceability system replacing the central database with blockchain. On the same direction in [21] the authors presented a blockchain agent-based simulator for cities in which the agents communicate via smart contracts. This solution takes advantage of the blockchain decentralisation and of its encryption mechanism that stores information in a secure way.

In [26] the authors explain the importance of developing blockchain-oriented software using smart contracts in a model-driven approach. In this scenario, BPMN is recognised as an enabler for process driven development of contracts. Also the authors in [5] present a conceptual model-driven approach based on BPMN choreographies, whose target platform relies on Hyperledger technologies. Similarly, in [22] the blockchain is recognised as a key enabler for the use of BPMN choreography diagrams as appropriate abstractions for top-down design of cross-organisational business processes. Finally, in [11] the authors explain the advantages of a decentralised blockchain solution for cross-organisational workflow management, highlighting the possibility to use the blockchain as an instrument for auditing of manual operations performed in relation to a process execution.

Among the works combining process management and blockchains, in a similar fashion with respect to our proposal, it is certainly worth to mention [30]. Here the authors propose the usage of smart contracts for controlling, monitoring and coordinating multi-organisational collaborative processes. Auditing results are then reported in a choreography model. 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 of providing details for each participating process, constitutes one of the main difference with respect to our proposal. Indeed, we consider choreography diagrams, which are more suitable in a multi-organisational context, where the internal details of a single process are generally not made available. In addition, the storage of more information on the blockchain infrastructure, as consequence of the considered greater level of detail, generally leads to a higher consumption of gas. In a similar way, in [28] the authors use collaboration diagrams to provide a framework permitting the execution of decentralised processes exploiting blockchain related technologies. Apart from the different kind of model used to represent the processes cooperation (collaboration vs. choreography), in this case the approach introduces a generic factory smart contract that will be reused for each process execution. In our opinion this could have some negative consequences, as it introduces a centralisation point possibly making the proposed infrastructure more susceptible to scalability and reliability issues. In addition, a single entity managing the information of many instances can be difficult to handle, possibly leading to information mixture among different instances. Differently, ChorChain generates a new contract for each choreography instance, resulting in a clearer separation of concern and a 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.

In [18] and [19] the Caterpillar tool is proposed. This is one of the first attempts to support the combination of business process management with a blockchain infrastructure. 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 the presented tool are presented in [16], where the authors propose a dynamic role binding model and a binding policy language for supporting collaborative business process. A similar extension of Caterpillar is proposed by the same authors in [17], where a list of components are provided for the update of models and their smart contracts at run-time, in order to react to unexpected situations during the execution. Lorikeet[29] is a similar tool, which however focusses more on the asset management and business process interactions on blockchain.

A similar direction is followed in [8, 9], where the authors translate BPEL processes into smart contracts. However, in this case the authors are mainly interested in investigating how to ensure data confidentiality in presence of an untrusted oracle. In [12], the authors provide an optimised execution method for business process contract generation based on Petri nets, while in [15] the authors propose an extension of the BPMN standard in order to include the representation of data in choreography models.

It is worth mentioning that the works reported above mainly, or only, focus on aspects related to the generation of the smart contract, while they generally overlook integration aspects related to the need of an infrastructure to support the whole life-cycle of choreographies and processes. Our work, instead, permits to derive a concrete implementation of choreography models, by relying on the underlying blockchain technology. The whole approach is encapsulated in a user-friendly framework that allows the developer to deal with all the phases of the choreography life-cycle, from the modelling to the deployment and execution. All these phases are supported by a web-based interface, easily accessible also to users not familiar with blockchain related technologies. Furthermore, ChorChain is the first tool implementing choreographies into smart contract following a model-driven approach and permitting their execution in a trustable way, also supporting natively the exchange of Ether represented as payments at the choreography level.

7 CONCLUSIONS AND FUTURE WORK

In this paper we tackled the problem of trust in the execution of distributed multiparty choreographies. Such an integration scenario has been advocated for several years by the service community. However, its concrete emergence has been impeded by the lack of a real infrastructure permitting to provide a sufficient level of trust

to the participants of a choreography in relation to the behaviour of the other participants in the same choreography.

The solution we propose in this work is a novel model-driven methodology for choreography-based systems, which relies on a blockchain infrastructure to enable the trustable execution of a choreography. The methodology is concretely supported by the ChorChain framework, which integrates mechanisms for the management of the whole choreography life-cycle, from choreography modelling to its execution. In particular, a relevant component of the proposed framework permits the translation of a choreography specification into a Solidity contract, once the participating partners have been identified. The smart contract is then deployed on the blockchain infrastructure and drives the choreography execution. Interactions are enabled according to the choreography specification, and when performed the details related to their occurrence are stored in the blockchain. The capability to observe the actual execution of the whole choreography reduces trust related issues.

ChorChain has been tested on the public permissionless Ethereum Testnet to get some indications on the possible costs resulting from the execution of a choreography, and to highlight possible scalability issues. The results we got are encouraging. Nonetheless, in the future we plan to work on further optimisations of the generated code to reduce the gas needed for the choreography execution. Moreover, it is worth mentioning that the proposed approach does not take into account possible issues related to privacy in relation to the information stored in the blockchain. This is certainly a relevant aspect to consider, and we intend to investigate it in the future. Possible solutions could consider the use of permissioned blockchain, or be based on existing tools like Hawk [14] or other proposals such as those introduced in [34, 35].

REFERENCES

[1] Aldazabal, A., Baily, T., Nanclares, F., Sadovykh, A., Hein, C., and Ritter, T. Automated model driven development processes. In *Model Driven Tool and Process Integration* (2008), Fraunhofer IRB Verlag, pp. 361 – 375.

[2] Ali, M., De Angelis, G., and Polini, A. Servicepot–an extensible registry for choreography governance. In *7th International Symposium on Service-Oriented System Engineering* (2013), IEEE, pp. 113–124.

[3] Almeida, S., Albuquerque, A., and Silva, A. An approach to develop software that uses blockchain. In *Software Engineering and Algorithms in Intelligent Systems* (2018), vol. 763 of *AISC*, Springer, pp. 346–355.

[4] Autili, M., Di Salle, A., Gallo, F., Pompilio, C., and Tivoli, M. Model-driven adaptation of service choreographies. In *33rd Annual ACM Symposium on Applied Computing* (2018), ACM, pp. 1441–1450.

[5] Autili, M., Gallo, F., Inverardi, P., Pompilio, C., and Tivoli, M. Introducing trust in service-oriented distributed systems through blockchain. In *International Workshop on Governing Adaptive and Unplanned Systems of Systems* (2019).

[6] Autili, M., Inverardi, P., and Tivoli, M. Choreos: Large scale choreographies for the future internet. In *2014 Software Evolution Week - IEEE Conference on Software Maintenance, Reengineering, and Reverse Engineering* (2014), pp. 391–394.

[7] Benyagoub, S., Ouederni, M., Aït-Ameur, Y., and Mashkoor, A. Incremental construction of realizable choreographies. In *NASA Formal Methods Symposium* (2018), vol. 10811 of *LNCS*, Springer, pp. 1–19.

[8] Carminati, B., Ferrari, E., and Rondanini, C. Blockchain as a platform for secure inter-organizational business processes. In *Collaboration and Internet Computing* (2018), IEEE, pp. 122–129.

[9] Carminati, B., Rondanini, C., and Ferrari, E. Confidential business process execution on blockchain. In *Web Services* (2018), IEEE, pp. 58–65.

[10] Corradini, F., Morichetta, A., Polini, A., Re, B., and Tiezzi, F. Collaboration vs. choreography conformance in BPMN 2.0: From theory to practice. In *EDOC* (2018), IEEE Computer Society, pp. 95–104.

[11] Fridgen, G., Radszuwill, S., Urbach, N., and Utz, L. Cross-organizational workflow management using blockchain technology - towards applicability, auditability, and automation. In *Hawaii International Conference on System Sciences* (2018), AIS Electronic Library, pp. 1–10.

[12] García-Bañuelos, L., Ponomarev, A., Dumas, M., and Weber, I. Optimized execution of business processes on blockchain. In *Business Process Management* (2017), vol. 10445 of *LNCS*, Springer, pp. 130–146.

[13] Hanada, Y., Hsiao, L., and Levis, P. Smart contracts for machine-to-machine communication: Possibilities and limitations. In *Internet of Things and Intelligence System* (2018), IEEE, pp. 130–136.

[14] Kosba, A., Miller, A., Shi, E., Wen, Z., and Papamanthou, C. Hawk: The blockchain model of cryptography and privacy-preserving smart contracts. In *IEEE symposium on security and privacy* (2016), pp. 839–858.

[15] Ladleif, J., Weske, M., and Weber, I. Modeling and enforcing blockchain-based choreographies. In *BPM* (2019), vol. 11675 of *LNCS*, Springer, pp. 69–85.

[16] López-Pintado, O., Dumas, M., García-Bañuelos, L., and Weber, I. Dynamic role binding in blockchain-based collaborative business processes. In *Advanced Information Systems Engineering* (2019), vol. 11483 of *LNCS*, Springer, pp. 399–414.

[17] López-Pintado, O., Dumas, M., García-Bañuelos, L., and Weber, I. Interpreted execution of business process models on blockchain. *ArXiv abs/1906.01420* (2019).

[18] López-Pintado, O., Dumas, M., and Weber, I. Caterpillar: A blockchain-based business process management system. In *BPM Demo Track and BPM Dissertation Award* (2017), vol. 1920, CEUR-WS.org.

[19] López-Pintado, O., García-Bañuelos, L., Dumas, M., Weber, I., and Ponomarev, A. CATERPILLAR: A business process execution engine on the ethereum blockchain. *CoRR abs/1808.03517* (2018).

[20] Madsen, M. F., Gaub, M., Høgnason, T., Kirkbro, M. E., Slaats, T., and Debois, S. Collaboration among adversaries: distributed workflow execution on a blockchain. In *Symposium on Foundations and Applications of Blockchain* (2018).

[21] Marrocco, L., Ferrer, E. C., Bucchiarone, A., Grignard, A., Alonso, L., Larson, K., et al. Basic: Towards a blockchained agent-based simulator for cities. In *Massively Multiagent Systems* (2018), vol. 11422 of *LNCS*, Springer, pp. 144–162.

[22] Mendling, J., Weber, I., and et al. Blockchains for business process management - challenges and opportunities. *ACM Transactions on Management Information Systems 9*, 1 (2018), 1–16.

[23] OMG. Business Process Model and Notation, 2011.

[24] Pastor, O. Model-driven development in practice: From requirements. In *Theory and Practice of Computer Science*, vol. 10139 of *LNCS*. Springer, 2017, pp. 405–410.

[25] Porru, S., Pinna, A., Marchesi, M., and Tonelli, R. Blockchain-oriented software engineering: challenges and new directions. In *Software Engineering Companion* (2017), IEEE/ACM, pp. 169–171.

[26] Rocha, H., and Ducasse, S. Preliminary steps towards modeling blockchain oriented software. In *Emerging Trends in Software Engineering for Blockchain* (2018), ACM, pp. 52–57.

[27] Schurr, P. H., and Ozanne, J. L. Influences on exchange processes: Buyers' preconceptions of a seller's trustworthiness and bargaining toughness. *Journal of consumer research 11*, 4 (1985), 939–953.

[28] Sturm, C., Szalanczi, J., Schönig, S., and Jablonski, S. A lean architecture for blockchain based decentralized process execution. In *Business Process Management Workshops* (2018), vol. 342 of *LNBIP*, Springer, pp. 361–373.

[29] Tran, A. B., Lu, Q., and Weber, I. Lorikeet: A model-driven engineering tool for blockchain-based business process execution and asset management. In *BPM Dissertation Award, Demonstration, and Industrial Track* (2018), vol. 2196, CEUR-WS.org, pp. 56–60.

[30] Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., and Mendling, J. Untrusted business process monitoring and execution using blockchain. In *Business Process Management* (2016), vol. 9850 of *LNCS*, Springer, pp. 329–347.

[31] Wessling, F., and Gruhn, V. Engineering software architectures of blockchain-oriented applications. In *Software Architecture Companion* (2018), IEEE, pp. 45–46.

[32] Wood, G. Ethereum: A secure decentralised generalised transaction ledger. Tech. rep., Ethereum Yellow Paper, 2014.

[33] Xu, X., Lu, Q., Liu, Y., Zhu, L., Yao, H., and Vasilakos, A. V. Designing blockchain-based applications a case study for imported product traceability. *Future Generation Computer Systems 92* (2019), 399–406.

[34] Zhang, F., Cecchetti, E., Croman, K., Juels, A., and Shi, E. Town crier: An authenticated data feed for smart contracts. In *Computer and Communications Security* (2016), ACM, pp. 270–282.

[35] Zyskind, G., Nathan, O., and Pentland, A. Decentralizing privacy: Using blockchain to protect personal data. In *Security and Privacy* (2015), IEEE Computer Society, pp. 180–184.