Trustworthy Cross-Organizational Collaborations with Hybrid On/Off-Chain Declarative Choreographies

Tiphaine Henry[1,3(✉)], Amina Brahem[2,4] , Nassim Laga[1], Julien Hatin[1], Walid Gaaloul[3], and Boualem Benatallah[5]

[1] Orange Labs, Paris, France
tiphaine.henry@orange.fr
[2] OASIS, University Tunis El Manar, Tunis, Tunisia
[3] Telecom SudParis, UMR 5157 Samovar, Institut Polytechnique de Paris, Palaiseau, France
[4] LIFAT, University of Tours, Tours, France
[5] University of New South Wales, Sydney, Australia

Abstract. Business Process Management communities increasingly adopt the blockchain technology to support trustworthy decentralized execution of processes. In this context, the interest in business process choreographies rises as they offer a distributed way to compose and control cross-organizational processes. In choreographies, the process view is distributed between participants to limit privacy leakages. Hence, the process observability (i.e., who knows what) is challenging. On one side, partners have no insight into each other's orchestration and communicate peer-to-peer via the public view. On the other side, they have to maintain their internal orchestrations' states consistent with the choreography's global state. The need to ensure a privacy-preserving method to enforce a blockchain-based execution thus rises. In the present work, we propose a unified solution for the hybrid on/off-chain generation and execution of business process choreographies. The public view, shared understanding of the cross-organizational process, is triggered by the on-chain smart contract. Participants generate their private views off-chain using this on-chain public view. They execute afterward the private views in their off-chain process execution engine. Our prototypical implementation demonstrates the feasibility of the approach.

Keywords: Decentralized choreographies · Business Process Management · Dynamic condition response graphs · Blockchain

1 Introduction

A cross-organizational process can be defined as a process scattered across different organizations. It comprises private processes carried out by individual partners, where internal data such as model and execution logs should not be visible to the other partners. It also includes a public process, where several

H. Hacid et al. (Eds.): ICSOC 2021, LNCS 13121, pp. 81–96, 2021.
https://doi.org/10.1007/978-3-030-91431-8_6

partners collaborate in a coordinated way. All partners should trust the execution state of the public process. A trade-off between ensuring the privacy of partners' private processes and the exposure of the public process thus arises. In cross-organizational processes, model flexibility is also at stake, as processes are dynamic: partners should be able to change their internal processes without impacting the public process [15]. Thus, the following question arises: *(RQ) how to carry out a separation of concerns that preserves the privacy of the private processes, trust of the public process, and flexibility of the whole?*

In the literature, business process choreographies answer the need for such separation of concerns by clearly specifying coordination tasks [1,6]. In addition, the public process is shared between participants to limit privacy leakages. Meanwhile, private views hold the set of (1) internal tasks of a particular partner not disclosed to the other partners, and (2) communication tasks in which this partner is involved, i.e., the projection of the public view over this partner [1]. However, the trustworthy execution of the public view remains challenging as it is often managed centrally [6].

Blockchain has been leveraged in the literature as a trustworthy coordination mechanism for collaborative business processes [5,6]. In [5], a smart contract manages the public workflow of an orchestration. However, in this approach, the execution of private tasks off-chain is only mentioned and the inner mechanism has not been detailed further. Additionally, in [6], the smart contract is used to manage the public view of a choreography, and so doing enforcing the order of messages. Nonetheless, in this work, a private/public separation is suggested but only the public view mechanism is implemented. Additionally, there is no on/off-chain enforcement of projections during deployment of the process instance. Thus, to the best of our knowledge, none of the retrieved works addresses the trustworthy deployment of choreographies. This deployment remains challenging as private information should not be shared between partners at design nor runtime. Moreover, none of the retrieved works proposes a detailed mechanism for the execution of projections using a hybrid on/off-chain mechanism.

In this paper, we contribute to the literature through a unified solution for the design and execution of business process choreographies in a hybrid on/off-chain fashion. The first contribution of this paper is a mechanism for the deployment of the global process which offers trustworthiness while preserving the separation of concerns. At deployment time, participants build incrementally the global process from a public view stored in a smart contract. Each participant will compute off-chain its role projection comprising public events where she is involved, and private events are kept off-chain for privacy concerns. This way, private control-flows remain in the participants' process engines, while blockchain systems ensure a tamper-proof public view. The blockchain has no access to the private events; it is the aggregation of all role projections that will render the global process. The second contribution is a hybrid on/off-chain mechanism for the execution of cross-organizational choreographies. The roles execute their internal tasks off-chain in their local process execution engine. Meanwhile, a smart contract manages public interactions. When the smart contract receives an interaction request initiated from one of the roles (sender or receiver(s)),

it executes the task and communicates its state back. The roles update their private states accordingly. Hence, we achieve a trustworthy separation of concerns preserving partners' private processes' privacy. Most existing works use an imperative paradigm such as BPMN. However, we chose to model choreographies with a declarative language that abstracts the control-flow through a set of rules or constraints [3,10], namely Dynamic-Condition-Response (DCR) graphs [11,25]. We believe that the declarative paradigm corresponds to the dynamic nature of choreography interactions, as business modelers cannot predefine all the execution paths of a model in constant evolution. Only essential constraints are specified in the model. We demonstrate our approach's feasibility through an implemented prototype and its effectiveness via a set of experiments.

The remainder of this paper is organized as follows. Section 2 introduces key concepts around blockchain and DCR graphs. Section 3 presents our motivating example. Section 4 details our approach. Section 5 presents an implemented prototype as a validation of our approach. Section 6 reviews the main known related work. Finally, Sect. 7 concludes the paper.

2 Background

A blockchain [26] is a distributed ledger holding a linked list of transactions organized in blocks. Each block contains (1) the reference to the previous block, (2) a tamper-evident digest of the transaction history to attest the integrity and blocks ordering, and (3) the list of the transactions to commit. Independent peers maintain the network. Peers use dedicated consensus algorithms such as proof-of-work or proof-of-stake to append transactions to the chain [13]. Some blockchains host smart contracts, deterministic scripts enforcing the terms of an agreement [14]. Business process approaches use blockchain to monitor in a decentralized fashion an agreed-upon scheme [17].

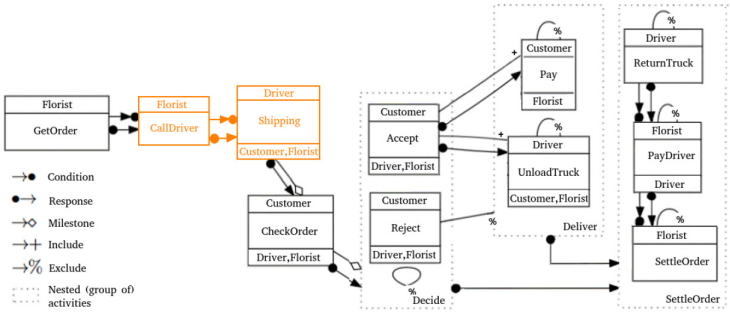
DCR is a declarative business process modeling language whose formalism is described in [11]. We refer to the following definition (cf [11]):

Definition 1. A DCR graph G is a tuple (E, M, L, f, $\longrightarrow \bullet$, $\bullet \longrightarrow$, $\longrightarrow \diamond$, $\longrightarrow +$, $\longrightarrow \%$), where:

- E is a set of events
- $M = (\text{In,Pe,Ex}) \subseteq E \times E \times E$ is a marking
- L is a set of labels
- $f : E \longrightarrow L$ is a labelling function
- $l \subseteq E \times E$ for $l \in \{\longrightarrow \bullet, \bullet \longrightarrow, \longrightarrow \diamond, \longrightarrow +, \longrightarrow \%\}$ are relations between events.

With DCR, processes are modelled as a set of *events* E linked together with *relations*.[1] Markings M capture the graph's state at runtime by referring to the triplet (currently included *events* In, currently pending *responses* Pe, previously

[1] A DCR event is equivalent to a BPMN activity.



(a) DCR graph of Flower Delivery *(in orange, a DCR subgraph)*

(b) Projection of the orange sub-graph over *Driver (Driver private view)*

(c) Projection of the orange sub-graph over *Florist (Florist private view)*

Fig. 1. DCR graph, and projections of a DCR graph chunk (in orange). (Color figure online)

executed *events* Ex). Relations model in a loosely fashion the constraints linking two *events*. The end-user can enact any enabled activity at any time and more than one time during a process instance execution. DCR graphs hold five types of relations. Two relations, *condition* and *milestone*, model pre-execution constraints. They restrain the enactment of an *event*. The *condition* relation implies that a task must be launched for another to start, while *milestone* requires full task completion. Three relations translate the effects of an *event* execution to the remaining activity markings. *Exclude* and *include* respectively lock or unlock the receiver task. *Response* sets the receiver task to pending upon completion of the source task. A *DCR choreography* [11,23] models and executes DCR graphs in a distributed way. It comprises choreography *events* that ease coordination between independent entities and internal events. We reconcile the definition of a DCR choreography proposed in [11] and formalize it as follows:

Definition 2. A DCR choreography is a triple (G, I, R) where G is a DCR graph, I is a set of interactions and R is a set of roles. An interaction i is a triple (e, r, r') in which the event e is initiated by the role r and received by the roles $r' \subset R \setminus \{r\}$. For an event $e \in E$, e.type is the type of the event, e.type $\in \{\epsilon, \gamma\}$, where (i) ϵ denotes the set of internal events in G, i.e., events having one initiator $r \in R$ and (ii) γ are the set of interactions in G ($\gamma = I$).

In Fig. 1a, *Shipping* is a choreography event sent by Driver and received by Florist and Customer. *GetOrder* is an internal event of the role Florist.

3 Motivating Example

Figure 1a represents a DCR choreography of a delivery process involving three participants: Customer, Florist, and Driver. Table 1 illustrates several executions of the graph instance. Each column corresponds to an event marking of the graph in the form (included, pending, executed). Each line stands for an event query triggered. For example, initially, no event is executed nor pending. The event *GetOrder* is included in the execution set. Thus the initial marking of GetOrder is (1, 0, 0). Each participant has control over the set of internal and choreography events where she is involved. We define this set of events as her private view. For example, the sub-graph in orange in Fig. 1a depicts the global view of a process involving three partners: Florist, Driver, and Customer. Figure 1b and Fig. 1c depict respectively the private views over Driver and Florist.

Requirements arise when dealing with the execution of such choreography. The activities for which some of the participants are not interested in (e.g., *ReturnTruck*) or confidential (e.g., *GetOrder*) must be kept private. The public view must express by design the information and requirements needed to execute the workflow. Moreover, public activities must be tamper-proof, and the execution flow fulfilled to keep on with the agreed-upon flow. The system must offer integrity by design. If a claim occurs, the system becomes the single source of truth. Former works on private and public views have been proposed before blockchain emergence [15,18]. A separation of concerns is reached by separating public and private views. However, trust in the execution of the public view is still needed. Blockchain brings two interesting properties with regards to our research: decentralization and tamper-proof logs. Thus, the public view of a business process could be completely decentralized by design while ensuring trust through the tamper-proof logs property. Nonetheless, two questions arise in this setting to preserve the separation of concerns between participants. The first question concerns the deployment of the global process in each local BPMS. The deployment shall not be managed by a centralized entity that would then upload the public view on-chain. Otherwise, the trust issue would rise again. Additionally, the question of how to ensure that projections are completed off-chain while avoiding any leakage of information remains. The second question concerns the execution of the global graph. The smart contracts acts as an entry-point to ensure correctness of execution of the public view. The mechanism managing the two-sided public/private execution of tasks needs to be defined to ensure that each participant can manage its projection in a trustworthy fashion.

4 The Approach

4.1 Design Time: Generating Public and Private Views

This section presents the hybrid on/off-chain protocol developed to generate the partners view-based projections (cf Fig. 2). A smart contract comprising (1) DCR execution constraints rules, and (2) a list of workflows initially empty, is used to manage workflow instances. The workflow responsible generates the new

Table 1. Evolution of the markings of the DCR graph in Fig. 1a

	Markings (included, pending, executed)							
	GetOrder	CallDriver	Shipping	CheckOrder	Accept	Reject	Deliver	SettleOrder
(init)	(1, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
GetOrder	(1, 0, 1)	(1, 1, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)
CallDriver	(1, 0, 1)	(1, 0, 1)	(1, 1, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)	(0, 0, 0)

workflow and updates the on-chain smart contract with the new public view. For each instance, the workflow comprises: the relation matrices and markings of the public view (cf. Sect. 2), the role addresses linked to each activity, and the IPFS hash of the textual input. The hash serves as a unique identifier for the workflow. Then, each participant computes its private view by combining the public view with its internal events. The output is a bitvectorized DCR graph. These private views constitute the entry point for the hybrid runtime execution. Finally, once the generation of role projections fulfilled, the smart contract unlocks the process instances for execution.

Used Formalism. Let (G, I, R) be a DCR choreography (cf. **Definition** 2), we define this DCR choreography through its public view G_γ and private views G_r, $\forall\, r \in R$, which are derived from G. We formalize G_γ and G_r, $\forall\, r \in R$ as follows:

Definition 3. Public View G_γ is a tuple $(E_\gamma, M_\gamma, L_\gamma, f_\gamma, \longrightarrow \bullet_\gamma, \bullet \longrightarrow_\gamma, \longrightarrow \diamond_\gamma, \longrightarrow +_\gamma, \longrightarrow \%_\gamma)$, where:

1. $E_\gamma = \{e \in I\}$
2. $M_\gamma = (In_\gamma, Pe_\gamma, Ex_\gamma)$ where $In_\gamma = In \cap E_\gamma$, $Pe_\gamma = Pe \cap E_\gamma$, and $Ex_\gamma = Ex \cap E_\gamma$
3. $f_\gamma(e) = f(e)$
4. $L_\gamma = img(f_\gamma)$
5. $\longrightarrow \bullet_\gamma = \longrightarrow \bullet \cap ((\longrightarrow \bullet\ E_\gamma) \times E_\gamma)$
6. $\bullet \longrightarrow_\gamma = \bullet \longrightarrow \cap ((\bullet \longrightarrow\ E_\gamma) \times E_\gamma)$
7. $\longrightarrow \diamond_\gamma = \longrightarrow \diamond \cap ((\longrightarrow \diamond\ E_\gamma) \times E_\gamma)$
8. $\longrightarrow +_\gamma = \longrightarrow + \cap ((\longrightarrow +\ E_\gamma) \times E_\gamma)$
9. $\longrightarrow \%_\gamma = \longrightarrow \% \cap ((\longrightarrow \%\ E_\gamma) \times E_\gamma)$

   Hence, $l_\gamma \in \{\longrightarrow \bullet_\gamma, \bullet \longrightarrow_\gamma, \longrightarrow \diamond_\gamma, \longrightarrow +_\gamma, \longrightarrow \%_\gamma\}$

Definition 4. Private Views For a role $r \in R$, $G_r =$ a tuple $(E_r, M_r, L_r, f_r, \longrightarrow \bullet_r, \bullet \longrightarrow_r, \longrightarrow \diamond_r, \longrightarrow +_r, \longrightarrow \%_r)$, where:

1. $E_r = \{e \in E \mid Initiator(e) = r \cup Receiver(e) = r\}$
2. $M_r = (In_r, Pe_r, Ex_r)$ where $In_r = In \cap E_r$, $Pe_r = Pe \cap E_r$, and $Ex_r = Ex \cap E_r$
3. $f_r(e) = f(e)$
4. $L_r = img(f_r)$

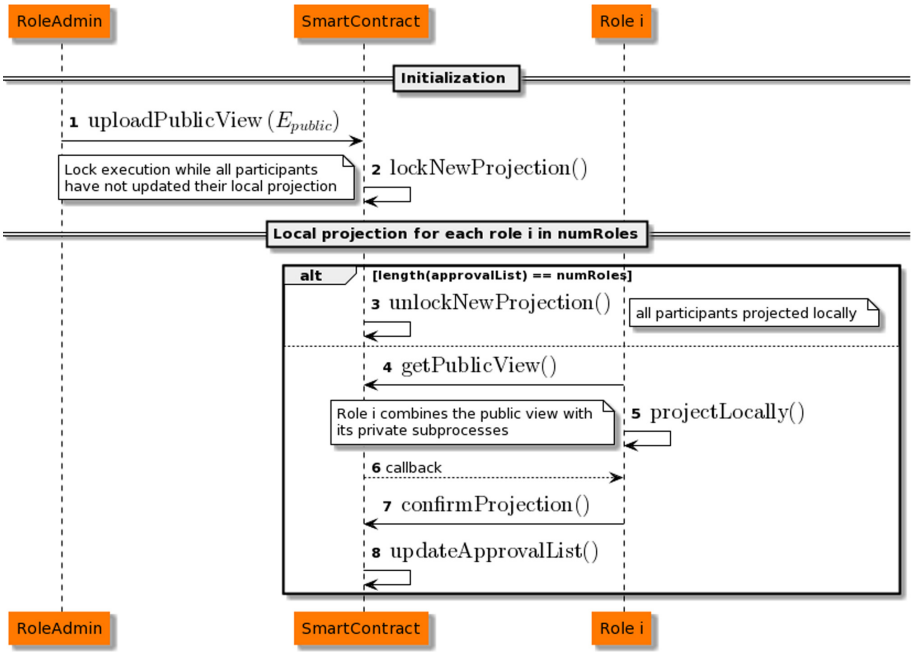


Fig. 2. Sequence diagram of the hybrid on/off-chain design protocol

5. $\longrightarrow \bullet_r = \longrightarrow \bullet \cap ((\longrightarrow \bullet\ E_r) \times E_r)$
6. $\bullet \longrightarrow_r = \bullet \longrightarrow \cap((\bullet \longrightarrow\ E_r) \times E_r)$
7. $\longrightarrow \diamond_r = \longrightarrow \diamond \cap ((\longrightarrow \diamond\ E_r) \times E_r)$
8. $\longrightarrow +_r = \longrightarrow + \cap ((\longrightarrow +\ E_r) \times E_r)$
9. $\longrightarrow \%_r = \longrightarrow \% \cap ((\longrightarrow \%\ E_r) \times E_r)$

   Hence, $l_r \in \{\longrightarrow \bullet_r, \bullet \longrightarrow_r, \longrightarrow \diamond_r, \longrightarrow +_r, \longrightarrow \%_r\}$

Translating DCR Graphs into Bitvectors. The public and private views are initially described as a textual input following the semantics prescribed in [16]. The reader can find input examples in the source code repository of our prototype.[2] We translate each view into a bitvector representation for execution in the off-chain and on-chain process execution engines [5,7]. We describe in the following paragraph the approach computing such representation.

The bitvector representation comprises (1) the five relation matrices of the DCR graph and (2) the three markings of the graph. The five relation matrices are computed out of an input view. For each relation $[event_i \longrightarrow event_j]$, the item a_{i_j} in the relation matrix is set to one. Besides, we generate the three initial bit-vector markings of the graph (Algorithm 1, l.3–5). The *executed* and *pending* initial markings are set to zero as no event has been executed yet.

[2] https://anonymous.4open.science/r/hybridChoreo-1CF8/.

Algorithm 1. Marking Vectorization of a private view

```
Data: G_r = (E, l)
Result: the list of included, executed, and pending marking vectors
1  Function initializeMarkings(E, l):
2      var len ← length(E);
       // INITIALIZE VECTORS
3      var In ← Vector(size : len);
4      var Pen ← Vector(size : len);
5      var Ex ← Vector(size : len);
       // DETECT INITIALLY INCLUDED EVENTS
6      var i=0;
7      forall the e ∈ E.ϵ do
8          var hasPreceedingEvent ← FALSE;
9          forall the rel ∈ l do
10             if rel.target == e then
11                 hasPreceedingEvent ← TRUE;
12                 break;
13         if not hasPreceedingEvent then
14             In[i] ← 1 ; // NO PRECEEDING EVENTS
15         i=i+1;
16     return [In, Pen, Ex]
17 End Function
```

The *included* state of the event is set to one if it has no pre-execution *condition* (Algorithm 1, l.6–15). We now illustrate the Florist projection bitvectorisation. First, we generate the five relation matrices. In the Florist private projection, a *condition* relation links *CallDriver* and *Shipping*. Thus, *Condition* $[id_{CallDriver}, id_{Shipping}] = 1$. The same protocol follows for each relation of the graph. We then compute the three markings of the projection. The *pending* and *executed* bit-vectors are filled with eleven zeros (one for each event of $E_{Florist}$). The Florist included bit-vector is filled similarly, except for *GetOrder* which is set to one (no pre-condition).

Hybrid On/Off-Chain Generation of Views. The generation of views comprises two steps: the on-chain public view first and private views.

The public view managed on-chain, G_γ, is the DCR graph consisting of the set of choreography events, i.e., events having one or many receivers and their relations, that model participants interactions. A representative of all participants first generates the approved bitvector representation of the public view (Fig. 2, step1). The public view consists of choreography events and their relations. Each role has a public blockchain address, and choreography events are mapped to a sender role. Moreover, the representative saves the textual public view input to IPFS to keep track of it, and saves the hash into the smart contract. The smart contract locks the process instance while waiting for each participant projection (Fig. 2, step2). A variable named *cnt*, initially set to zero, keeps track

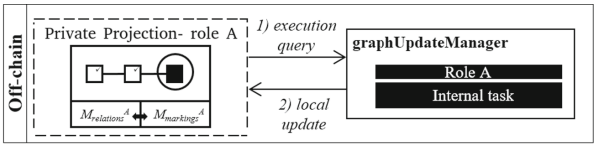
of the number of projections realized. The process instance is unlocked for execution when *cnt* equals the number of participants. The public events of Fig. 1a are {*Shipping, CheckOrder, Accept, Reject, Pay, UnloadTruck, PayDriver*}. The smart contract stores these events and relations where at least two public events are involved. Internal events such as {*ReturnTruck*} for Driver, or {*GetOrder*, *CallDriver*, *SettleOrder*} for Florist are kept off-chain.

Once the public view populates the smart contract, each participant fetches it (Fig. 2, step4). The private projection is generated by extracting all the events of G_γ where the participant is an initiator or a receiver in a choreography event. We conjointly extract relations connecting these events. Afterward, the participant combines off-chain the public view with its internal events (Fig. 2, step4). The obtained projection over the role r is G_r. A dedicated smart contract function named, *confirmProjection()*, enables participants to update *cnt* after the local projection. The function uses two mapping variables. The first mapping, *approval*, records whether a participant has generated its local projection. The second mapping, *didFetch*, records whether the participant did fetch the public view (necessary condition to realize the projection). The following constraints restrain *cnt* update: (i) the sender's address must belong to the list of addresses white-listed in the smart contract, (ii) participants can only update the variable once, and (iii) must have fetched the public projection first. In the motivating example, Florist asks the public projection to the smart contract. The smart contract verifies that its address belongs to the white-list, forwards the public view to Florist, and updates *cnt* to 1. Florist projects the view over her role. She obtains a set of receive events: {*Shipping, CheckOrder, Accept, Reject, Pay, UnloadTruck*}, and one send event {*PayDriver*}. She then adds its internal activities {*GetOrder, CallDriver, SettleOrder*} to the projection. Lastly, Florist triggers $confirmProjection()$.

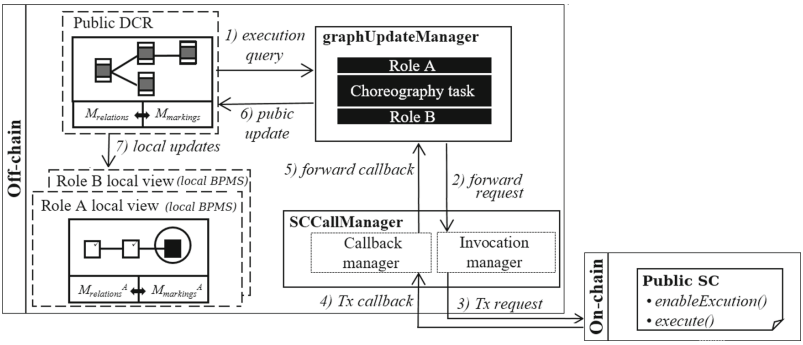
4.2 Hybrid Off/On-Chain Runtime Execution

Our approach proposes a hybrid execution at runtime: the private DCR execution engine of the involved participants manages the private projections. Meanwhile, a smart contract called S triggers the execution logic of the public tasks on blockchain. An event execution query comprises the name of the event and its class: *internal*, or *choreography*. The execution logic depends on the event class. The private and public projections communicate via choreography events. Participant executes private events off-chain (cf. Fig. 3a). For an internal event, the private process engine looks at its private markings (see Fig. 3a). If the event is enabled,[3] we apply post-execution constraints to the bound events (i.e., events are set to pending, included, or excluded), and update the marking accordingly. For example, the execution request of *GetOrder* (Fig. 1a) will succeed: it does not have any pre-execution constraint. Thus, the executed marking of the event *GetOrder* will be set to one. The post-execution constraints (*condition* and *response*) will unlock *CallDriver* and set its pending marking to one.

[3] An event is *enabled* if the following preconditions are fulfilled: the event is included, and the *condition* and *milestone* relations are executed.



(a) Execution of an internal event



(b) Execution of a choreography event

Fig. 3. The execution scheme logic of DCR choreography events

The smart contract S handles the execution of the choreography send and receive events (cf. Fig. 3b). S holds the bitvector representation of the public view and two functions: *enableExecution()* checks the enabling preconditions, and *execute()* computes the enabled event and updates the marking vectors. The execution of a choreography event follows the subsequent steps (see Fig. 3b). First, the backend receives an execution query (step 1) and forwards it to the smart contract API (step 2). The latter sends a transaction (Tx) to S to call the function *enableExecution()* (step 3). The Tx includes the event's name to execute, the event initiator, the receiver (if it is a choreography event), and the event state (enabled, included, executed). If the activation conditions are verified, the function *execute()* updates the event state (the three bit-vectors) and the public projection state (the five relation matrices). The Tx callback containing the updated states is sent back to the smart contract API (step 4), which forwards it to the local backend (step 5). The backend updates the public projection (step 6). Changes are propagated to the concerned private projections (step 7). Choreography events are by nature of interest to process participants. S makes their execution management trustworthy as its behavior is deterministic, and the choreography states stored into the smart contract are tamper-proof.

5 Evaluation

Our proof of concept is a hybrid on/off-chain business process engine managing declarative choreographies (code repo: cf footnote 2). We use a Ganache

testnet to deploy the public smart contract S which manages each process. S comprises (1) execution constraints rules, and (2) a list of workflows initially empty. The initial cost of deployment of S is 0.06413472 ETH (i.e., 137.6$). For each workflow, RoleAdmin (1) generates the public view bitvector representation (Sect. 4.1), (2) saves the textual public view input to IPFS, and (3) registers the new workflow on-chain by calling the function uploadPublicView. The workflow is identified by the IPFS unique hash. Participants interact with the smart contract via API calls to generate their private views. Afterwards, the process instance is released for execution. The local process execution engine executes internal events off-chain and forwards choreography events to the blockchain.

We instantiate three cross-organizational processes in the platform to assess the execution cost in terms of gas fees and time. We test two workflows from the literature: the invoice and oncology workflows [25], and the motivating example. We run the experiments on a personal computer with an Intel i5 core CPU, 4 GB of RAM. At the time of writing, 1ETH = 2,145.73$. We evaluate the public-to-private projection costs of the system for the deployment of the three processes mentioned above (cf. Table 2a). For each workflow, the public view registration cost is worth 0.068352 ETH (146.7$) for the delivery workflow, 0.040947 ETH (87.9$) for the invoice workflow, and 0.065019 ETH (139.5$) for the oncology workflow. Afterwards, each role fetches the public view, and confirms its projection. The delivery and invoice workflows share the same costs for fetching the public view and confirming the projection. Such cost, corresponding to updating *approval* and $didFetch$, is proportional to the number of roles registered. The total cost for instantiating a choreography corresponds to public view upload, and the number of roles $\#R$ times the private projection cost. It is worth 0.078534 ETH (168.5$), 0.051129 ETH (109.7$), and 0.079795 ETH (171.2$) for the delivery, invoice, and oncology workflows respectively. The public-to-private total projection cost depends on the number of roles and events.

We also evaluate the performance of the system at runtime: Table 2a presents the results obtained after the enactment of one trace. The reported execution time factors the transaction confirmation time. The average transaction fees requested for a task execution are smaller than the process instantiation ones. Moreover, the average execution time for a private task is one order of magnitude smaller than the one needed for a public task. Indeed, we compute private activities off-chain. Thus the execution time of a private event corresponds comprises checking the event nature (private or public), and updating private markings. On the opposite, the execution of public activities comprises an interaction with the blockchain network. Against this backdrop, the local execution of private tasks reduces the overall execution time.

Finally, we compare the transaction costs of our approach to the BPMN-based experiments presented in [8]. We translate into DCR choreographies the two open-sourced BPMN choreographies presented in [5], namely *supply chain* and *incident management*. We deploy and execute the choreography in our prototype, and compare the results. Table 3 shows the instantiation and task execution average gas fees; task execution fees correspond to the average cost of

Table 2. Hybrid on/off-chain Projection and execution costs Hybrid on/off-chain Projection and execution costs

(a) Public-to-private projection costs, *W.* =*Workflow*

Step	**Role**	**Function**	**Delivery W.**	**Invoice W.**	**Oncology W.**
A	RoleAdmin	*uploadPublicView()*	0.068352 ETH	0.040947 ETH	0.065019 ETH
B1	Role r in *R*	*fetchPublicView()*	0.002006 ETH	0.002006 ETH	0.002139 ETH
B2	Role r in *R*	*confirmProjection()*	0.001388 ETH	0.001388 ETH	0.001555 ETH
Total Cost = A + #*R*.(B1+B2)			0.078534 ETH	0.051129 ETH	0.079795 ETH

(b) Task execution costs *(Pub/Pri= public/private tasks).*

Workflow					**Tx. Fees**	**Exec Time**	
Name	*#Parties*	*#Pub*	*#Pri*	*#Constraints*	*Task Exec*	*Pub*	*Pri*
Delivery	3	9	1	28	0.0093 ETH	15s	1s
Invoice	3	8	2	15	0.0069 ETH	10s	1s
Oncology	4	10	3	21	0.0117 ETH	19s	2s
				Mean	0.0093 ETH	14.6s	1.3s

Table 3. Gas fees comparison of BPMN [8] and DCR choreographies (our approach) run on the Ethereum blockchain

Workflow	#Tasks	#Gateways	Gas fees	[8] (BPMN)	Our approach
Supply chain [5]	10	2	Instantiation	1,100,590	1,074,178
			Task exec.	566,861	478,527
Incident Mgt. [5]	9	6	Instantiation	1 119,803	930,399
			Task exec.	324,420	456,887

execution of a task. A gain of 26,412 gas for the supply chain workflow, and 189,404 gas for the incident management workflow can be noticed with the DCR approach. Thus, the DCR-based smart-contract requires less fees for instantiation than the BPMN one in these workflows. Regarding task execution costs, the modeling choice does not seem to impact gas fees: a gain can be noticed with DCR in the supply chain workflow, but not in the incident management one. The number of gateways (2 in the supply chain, and six in the incident management workflow) may explain such disparity. Indeed, each exclusive gateway is translated into an include and a response relation for each decision path in the DCR model. Such translation may explain the gas difference.

6 Related Work

Regarding traditional view-based approaches, authors in [15,18] use process views to build an abstracted version of each partners' private processes in order to hide its internal structure. In [18], authors define a SOG (Symbolic Observation Graph) for each choreography participants. A SOG is an abstraction of

the reachability state graph of a formally modeled process (e.g., an LTS). The nodes in the SOG are meta-states, i.e., a set of states connected by unobserved (internal) activities, and the edges are labelled with observed (interaction) activities. The SOG of the choreography process is the product of the SOGs of the participants. In [15], roles inter-connect via a set of virtual activities. These virtual activities abstract choreography interactions, and are enacted by a trusted third-party. In these works, partners' privacy is reached by separating public and private views. However, trust issues remain as shared execution logic and data are managed in a centralized fashion, often by a third-party [6].

Blockchains have been leveraged as trusted mechanisms to ensure the public view correctness in recent work. In the following, we classify related works managing collaborative processes on-chain according to (1) the choice of paradigm which impacts the system flexibility and scalability, (2) the public/private views separation which impacts confidentiality, and (3) the deployment which impacts participants trust. The **paradigm** criterion refers to the process modeling choice used to represent collaborative processes on blockchain. In [4–6,19,21,24], the imperative modeling approach is chosen: BPMN business models describe the control flow in a sequential manner. Other works such as [2,7,9,12,20] use the *declarative modeling* approach where only execution constraints are specified. [9,12] propose LTL for smart contract parametrized pre and post-execution conditions, however without including implementations. Authors in [20] use the artifact-centric language, in [2] XML, and in [7] DCR. The **view-based** criterion refers to the separate display of the global process: in a view-based setting, participants only have access to their tasks. [2,4,5,7,9,12,20,21,24] do not consider the public/private view separation. For example, in [5,7,19], authors handle orchestration schemes only. [6] considers a choreography but authors do not expand on the participants' private workflows execution and deployment. Though the generation of the public and private views in [6] is suggested, projections are not enforced in a trustworthy fashion in this work. The **deployment** criterion refers to the deployment model chosen for collaborative processes. *Regarding fully on-chain schemes*, a translator maps directly BPMN [5,6,19], DCR [7], or XML [2] models into Solidity. Additionally, a custom interface binds local execution engines with blockchain in [6]. In [24], authors run choreographies with Bitcoin instead of smart contracts. [9,12] advise the direct end-to-end deployment of public processes. [20] stores the hash of an artifact-based multi-party process in a smart contract but no details are given on off-chain tasks. *Regarding hybrid on/off-chain schemes*, [21] proposed a set of on/off-chain connectors, but processes are intra-organizational and the system allows only monetary operations. In [4], a gateway enables interactions of an off-chain intra-organizational BPMN process with heterogeneous blockchains.

Most blockchain-based collaborative processes cited in the literature do not consider declarative choreographies. When they do, they do not distinguish the partners' internal processes and the public view of the choreography when deployed to the chain. Consequently, the contribution of this paper is to answer cross-organizational needs for process flexibility and trustworthy separation of

concerns. To do so, we build a collaborative BPMS that offers modeling flexibility, as well as a trustworthy and privacy-preserving separation of concerns. We chose a declarative language that offers collaboration flexibility, necessary due to the dynamic nature of collaborations. We add to this design choice the public/private view separation and a hybrid deployment to enforce in a trustworthy fashion the separation of concerns.

7 Discussion and Conclusion

This paper leverages the management of business process choreographies using blockchain to address the need for a trustworthy separation of concerns. Additionally, we model choreographies with a declarative language called DCR. This language offers loosely-constrained models to meet the flexibility requirements of cross-organizational processes. To enhance privacy at design time, the public view of the choreography is stored in a smart contract, and participants generate their private view off-chain. On the execution side, internal events are executed locally for privacy concerns, while choreography events are executed on-chain for accountability concerns.

This approach represents a first effort to separate the public and private views of a declarative choreography and proceed with its hybrid off/on-chain management. Results confirm the advantages of separating public from private events to ensure privacy while leveraging blockchain as a decentralized execution infrastructure. Moreover, the local execution of private events leads to time and economic gains. Our approach works if there is no public event. Then, no public projection is generated. Multi-instance choreographies are also possible: for each new instance, a workflow instance is added to the smart contract. Besides, experiments on graphs of alternative complexity (be it the number of participants or activities) should confirm preliminary results. A limitation to our approach concerns the public/private exchange of information. In our setting, the information published in the smart contract is public. Consortium or private blockchains, coupled to off-chain oracles to exchange sensitive information with the smart contract, could answer privacy concerns. Furthermore, we rely on the truthfulness of participants to execute their private projections and do not ensure the correct enforcement of private processes. This concern, inherent to choreographies, is part of ongoing research efforts.

As future work, we plan to use side channels [22] to manage on-chain process instances to save transaction costs and reduce task execution latency. Only two blockchain transactions would be of need: one to instantiate the process execution channel, and one to settle it. Additionally, a need rises regarding the ability of participants to change the global workflow at runtime. An avenue for future work is to propose such functionality to the proposed system, building on the declarative paradigm to define flexibly authorizations and obligations.

References

1. van der Aalst, W.M.P., Weske, M.: The P2P approach to interorganizational workflows. In: Dittrich, K.R., Geppert, A., Norrie, M.C. (eds.) CAiSE 2001. LNCS, vol. 2068, pp. 140–156. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45341-5_10
2. Brahem, A., et al.: Blockchain's fame reaches the execution of personalized touristic itineraries. In: WETICE, pp. 186–191. IEEE (2019)
3. Fahland, D., Mendling, J., Reijers, H.A., Weber, B., Weidlich, M., Zugal, S.: Declarative versus imperative process modeling languages: the issue of maintainability. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (eds.) BPM 2009. LNBIP, vol. 43, pp. 477–488. Springer, Heidelberg (2010). https://doi.org/10.1007/978-3-642-12186-9_45
4. Falazi, G., et al.: Process-based composition of permissioned and permissionless blockchain smart contracts. In: EDOC (2019)
5. Weber, I., Xu, X., Riveret, R., Governatori, G., Ponomarev, A., Mendling, J.: Untrusted business process monitoring and execution using blockchain. In: La Rosa, M., Loos, P., Pastor, O. (eds.) BPM 2016. LNCS, vol. 9850, pp. 329–347. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-45348-4_19
6. Ladleif, J., Weske, M., Weber, I.: Modeling and enforcing blockchain-based choreographies. In: Hildebrandt, T., van Dongen, B.F., Röglinger, M., Mendling, J. (eds.) BPM 2019. LNCS, vol. 11675, pp. 69–85. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-26619-6_7
7. Madsen et al., M.F.: Collaboration among adversaries: distributed workflow execution on a blockchain. In: FAB, p. 8 (2018)
8. López-Pintado O., et al.: CATERPILLAR: a business process execution engine on the ethereum blockchain. Softw.: Pract. Exp. **49**(7), 1162–1193 (2019)
9. Hull, R., Batra, V.S., Chen, Y.-M., Deutsch, A., Heath III, F.F.T., Vianu, V.: Towards a shared ledger business collaboration language based on data-aware processes. In: Sheng, Q.Z., Stroulia, E., Tata, S., Bhiri, S. (eds.) ICSOC 2016. LNCS, vol. 9936, pp. 18–36. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-46295-0_2
10. Goedertier, S., et al.: Declarative business process modelling: principles and modelling languages. Enterp. Inf. Syst. **9**(2), 161–185 (2015)
11. Hildebrandt, T.T., Slaats, T., López, H.A., Debois, S., Carbone, M.: Declarative choreographies and liveness. In: Pérez, J.A., Yoshida, N. (eds.) FORTE 2019. LNCS, vol. 11535, pp. 129–147. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21759-4_8
12. de Sousa et al, V.A.: B-MERODE: a model-driven engineering and artifact-centric approach to generate smart contracts. In: CAiSE (2020)
13. Bach, L., Mihaljevic, B., Zagar, M.: Comparative analysis of blockchain consensus algorithms. In: MIPRO, pp. 1545–1550. IEEE (2018)
14. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. White paper, vol. 3, no. 37 (2014)
15. Chebbi, I., Dustdar, S., Tata, S.: The view-based approach to dynamic inter-organizational workflow cooperation. Data Knowl. Eng. **56**(2), 139–173 (2006)
16. Debois, S., Hildebrandt, T.: The DCR Workbench: Declarative Choreographies for Collaborative Processes, pp. 99–124. River Publishers (2017)
17. Henry, T., Laga, N., Hatin, J., Gaaloul, W., Boughzala, I.: Cross-collaboration processes based on blockchain and IoT: a survey. In: HICSS (2021)

18. Klai, K., Tata, S., Desel, J.: Symbolic abstraction and deadlock-freeness verification of inter-enterprise processes. In: Dayal, U., Eder, J., Koehler, J., Reijers, H.A. (eds.) BPM 2009. LNCS, vol. 5701, pp. 294–309. Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-642-03848-8_20
19. López-Pintado, O., Dumas, M., García-Bañuelos, L., Weber, I.: Dynamic role binding in blockchain-based collaborative business processes. In: Giorgini, P., Weber, B. (eds.) CAiSE 2019. LNCS, vol. 11483, pp. 399–414. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-21290-2_25
20. Meroni, G., Plebani, P., Vona, F., et al.: Trusted artifact-driven process monitoring of multi-party business processes with blockchain. In: Di Ciccio, C. (ed.) BPM 2019. LNBIP, vol. 361, pp. 55–70. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30429-4_5
21. Palacin, L.: Accelerate blockchain technology adoption with Bonita BPM and Chain Core, pp. 04–08 (2018)
22. Papadis, N., Tassiulas, L.: Blockchain-based payment channel networks: challenges and recent advances. IEEE Access **8**, 227596–227609 (2020)
23. Peltz, C.: Web services orchestration and choreography. Computer **36**, 46–52 (2003)
24. Prybila, C., Schulte, S., Hochreiner, C., Weber, I.: Runtime verification for business processes utilizing the bitcoin blockchain. FGCS **107**, 816–831 (2020)
25. Slaats, T., Hildebrandt, T.T., Carbone, M., Völzer, H.: Flexible process notations for cross-organizational case management systems. ITU Copenhagen (2015)
26. Underwood, S.: Blockchain beyond bitcoin. ACM **59**(11), 15–17 (2016)