Towards a trustworthy and adaptive execution of business process choreographies

Amina Brahem, Tiphaine Henry, Sami Bhiri, Thomas Devogele, Nizar Messai, Yacine Sam, Walid Gaaloul

Towards a trustworthy and adaptive execution of business process choreographies

Amina Brahem*[1,2,4],Tiphaine Henry†[4], Sami Bhiri‡[3], Thomas Devogele§[1], Nizar Messai¶[1], Yacine Sam‖[1], Walid Gaaloul**[5]**biography not provided by author's choice.**
Email: * amina.brahem@univ-tours.fr, †tiphaine.henry@cea.fr, §¶‖firstname.lastname@univ-tours.fr,
‡sami.bhiri@gmail.com, ** walid.gaaloul@telecom-sudparis.eu
[1]*LIFAT, University of Tours, Tours, France*
[2]*OASIS, National Engineering School of Tunis, University Tunis El Manar, Tunis, Tunisia*
[3]*Labtim, University of Monastir, Monastir, Tunisia*
[4]*Université Paris-Saclay, CEA, List, F-91120, Palaiseau, France*
[5]*Telecom SudParis, UMR 5157 Samovar, University Paris-Saclay, Paris, France*

***Abstract*—Blockchain technologies have emerged to serve as a trust basis for the monitoring and execution of business processes, particularly business process choreographies. However, dealing with changes in smart contract-enabled business processes remains an open issue. For any required modification to an existing smart contract (SC), a new version of the SC with a new address is deployed on the blockchain and stored in a contract registry. Moreover, in a choreography, a change in a partner process might affect the processes of other partners, and thus, must be propagated to partners affected by the change. In this paper, we propose an approach overcoming the limitations of SCs and allowing for the change management of blockchain-enabled declarative business process choreographies modeled as DCR graphs. Our approach allows a partner in a running blockchain-based DCR choreography instance to change its private process. A change impacting other partners is propagated to their processes in a decentralized manner using a SC. The change propagation mechanism ensures the compatibility checks between public processes of the partners and the consistency between the private and public processes of one partner. We demonstrate the approach's feasibility through an implemented prototype and its effectiveness via a set of evaluation tests.**

***Index Terms*—Process choreography, Change propagation, DCR graph, Smart contract, Dynamic change correctness.**

I. Introduction

Blockchain technologies have emerged to serve as a trust basis for the monitoring and execution of business processes [1], and particularly business process choreographies [2]. This is due to several mechanisms, be it the consensus method applied among the nodes to validate a transaction, the immutable nature of transactions, process automation using smart Contracts (SCs for short) and its ability to manage decentralized, peer-to-peer interactions [3].

Both imperative and declarative process modelling paradigms have been used to deal with blockchain-based business process execution. Proposed techniques include translation of BPMN collaboration models into SCs [2] and execution engines of declarative orchestration processes called Dynamic-Condition-Response (DCR for short) graphs [4]. However, dealing with changes in blockchain-enabled business processes remains an open research issue [5].

Business processes managed by "static" SCs cannot be upgraded because the SCs are immutable once deployed. Efforts exist to support versioning in SCs [6]. For any required modification on the existing SC, a new version of the SC with new address is deployed on the blockchain and stored in a contract registry. So the process may have new version and in future interactions should be consistent with it. However, with SC having many versions, it is difficult to maintain inter-dependent SCs links and to copy data from old to new version of the contract [6]. Moreover, these are all costly operations.

We aim to enable a way to integrate change management into SCs implementing the business logic of process choreographies without deploying them again. This circumvents the aforementioned problems related to versioning.

In a choreography, each partner manages its private process and interacts with other partners via its public process. The model comprising all interactions is called choreography process [7], [8]. In a running choreography instance, a change may consist of a simple change operation (ADD/REMOVE/UPDATE) or combination of change operations [7], [9]. A change in the instance of a partner process may affect other partners' process instances. Hence, change must be propagated to the affected partners of the choreography instance [7], [10]. In a trip e-booking process, for example, a hotel may close its catering facility for reparations and thus DELETE the dining service. A Tourist having booked the hotel with dinner included will be unable to reach the hotel service "ProvideDinner". Additionally, a new restaurant may want to establish (ADD) a convention with the hotel. This new relationship will affect the Tourist interested in trying the restaurant. Thus, the ADD change must be propagated to the Tourist process instance (further details of the mentioned scenario can be found in the running example in Section. II).

By adopting the three levels of granularity: (i) choreography, (ii) public and (iii) private processes, change propagation must be considered at three levels: i) from private to public process of the change initiator, ii) between partners public processes and iii) from public to private processes of the partners [7]. One change we consider in this work is, as we

said previously, a simple (ADD/REMOVE/UPDATE) change operation or combination of these change operations.

One can assure that the choreography is correct-by-construction when applying a change where no deadlocks or livelocks occur [11]. However, the propagation of the change must ensure the deadlock and livelock freedom while ensuring at the same time the correctness criteria which consist of (i) the compatibility and (ii) consistency checks [7], [12], [13].

To ensure compatibility, one has also to guarantee that neither the structural nor behavioral compatibility of partners processes are violated after a change [7], [12], [14], [15]. Structural compatibility checks consist of ensuring that there is at least one potential *send* message assigned to a partner with a corresponding *receive* message assigned to another partner [15]. Behavioral compatibility refers to ensuring that the choreography process after the change is safe and terminates in acceptable state. In other words, no deadlocks should occur between partners public processes during the choreography execution after the change [7], [14]. For example, in the trip e-booking process, the task "HaveDinner" is a public task. It is composed of two messages, namely the send and receive messages, that are respectively assigned to Tourist and NewRestaurant. When NewRestaurant DELETES the receive message "HaveDinner", a structural incompatibility occurs as the corresponding send message is still present in the Tourist process. It also leads to behavioral incompatibility as the process encounters a deadlock and will be unable to continue its execution.

Moreover, in business process choreographies, one partner's private process has to stay consistent with its corresponding public process after the change. Expressly, if the effect of one (or many) change(s) over one partner's public process leads to a live and safe process, then applying the effect of this change on the partner's private process will lead to a safe and live private process as well.

To the best of our knowledge, the change propagation in blockchain-based declarative choreographies management systems, and especially proving the correctness of the change propagation in such choreographies has not been studied yet. Therefore, in this paper we focus on the following research question: *(RQ) How to ensure correctness while propagating a change in declarative blockchain-based choreography processes?*

In the present paper, we propose a change mechanism to bring adaptiveness to the trustworthy execution of declarative choreographies while ensuring correctness of the change propagation. Declarative choreographies are modeled using the declarative language called DCR graphs. With DCR, processes are modelled as a set of events linked together with relations (a kind of temporal dependencies) [11], [16].

In this context, we propose in our previous work in [17] an approach for trustworthy propagation of changes in declarative business process choreographies modeled as DCR graphs. The approach in [17] allows a partner in a running DCR choreography instance to change its private process. The change effect is then propagated to the processes of the partners affected by the change. Changes affecting private activities are applied off-chain while changes impacting interactions with other partners are managed on-chain through the SC.

However, we do not demonstrate the correctness of the change propagation between the partners public processes neither between the public and private processes of one partner. Moreover, we focus only on the running instance of the choreography and do not precise what will happen to future running instances after applying the change.

The present paper reviews and extends the previous work in [17]. First, we give precisely the different properties which are necessary and sufficient to guarantee the correctness criteria: (i) structural and behavioral compatibility between partners public processes, and (ii) consistency between one partner's private and public process. Second, we consider evolutionary changes. Evolutionary changes are changes that affect the structure of the process in order to improve efficiency or responsiveness. In our work, the change is applied starting from the running instance of the choreography. New instances are handled according the most recent version of the choreography, i.e., the choreography after the change.

The remainder of this paper is organized as follows. Section II presents fundamental definitions used in the approach and introduces an illustrating example. SectionIII describes the proposed approach. Section IV gives the proof of change propagation correctness. Section V presents the implemented prototype and some evaluation tests. Section VI reviews the main known related work. Finally, Section VII presents the conclusions and insights into future work.

II. Basic concepts and illustrating example

Dynamic-Condition-Response (DCR) is one of many declarative BP modeling languages. Its formalism is is described in [16]. We refer to the following definition (cf [16]):

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 language, 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 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* $\longrightarrow \bullet$ and *milestone* $\longrightarrow \diamond$, 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*

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

execution to the remaining activity markings. *Exclude* $\longrightarrow \%$ and *include* $\longrightarrow +$ respectively lock or unlock the receiver task. *Response* $\bullet \longrightarrow$ sets the receiver task to pending upon completion of the source task.

For example, the *condition* relation from *MoveToSite* to *PurchasePass* in Fig. 1 implies that *MoveToSite* must have been launched, not necessarily terminated, so that *PurchasePass* can start. The *milestone* relation from *PurchasePass* to *ProvideAccomodation* implies that *PurchasePass* must be finished for *ProvideAccomodation* to start. Three relations translate the effects of an *event* execution to the remaining activity markings. The *inclusion* relation from *ProvideAccomodation* to *ProvideDinner* states that the execution of *ProvideAccomodation* unlocks *ProvideDinner*. On the opposite, the *exclusion* relation from *VisitCastleWithGuide* to itself states that the enactment of this activity happens just once. Finally, the *response* relation from *MoveToSite* to *PurchasePass* sets *PurchasePass* to pending when *MoveToSite* is executed (i.e., *PurchasePass* is started and waiting for completion).

A **DCR choreography** models interactions between partners. Its execution is done in a distributed way. A DCR choreography is defined as follows:

Definition 2. A **DCR choreography** C 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 $r' \subset R \setminus \{r\}$.

Which leads us to the definitions of one partner's **public** and **private** DCR processes.

Definition 3. A **DCR public process** , G_r of one partner represents the **projection** of the DCR choreography over this partner. For a role $r \in R$, G_r is 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) $E_r = \{e \in E \mid Initiator(e) = r \cup Receiver(e) = r\}$ is the set of events involving r
2) $M_r = (In_r, Pe_r, Ex_r)$ where $In_r = In \cap E_r$, $Pe_r = Pe \cap E_r$, and $Ex_r = Ex \cap E_r$ is a marking of an event in E_r
3) $L_r = img(f_r)$ is the set of labels of events in E_r
4) $f_r(e) = f(e)$ is the labelling function of events in E_r
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, $l_r \in \{\longrightarrow \bullet_r, \bullet \longrightarrow_r, \longrightarrow \diamond_r, \longrightarrow +_r, \longrightarrow \%_r\}$ are relations between events in E_r.

Definition 4. A **DCR private process**, P_r of one partner r is a kind of a refinement of the public DCR process, i.e., it comprises the public interactions in addition to the internal events related to this partner. We note ϵ_r the set of internal events related to one role r. ϵ_r describes the behavior of role r that is not visible to other partners. Thus, for a role $r \in R$, a DCR private process P_r is a tuple (E_{Pr}, M_{Pr}, L_{Pr}, f_{Pr}, $\longrightarrow \bullet_{Pr}$, $\bullet \longrightarrow_{Pr}$, $\longrightarrow \diamond_{Pr}$, $\longrightarrow +_{Pr}$, $\longrightarrow \%_{Pr}$), where:

1) $E_{Pr} = \{E_r \cup \epsilon_r\}$ is the set of interactions and internal events involving r
2) $M_{Pr} = (In_{Pr}, Pe_{Pr}, Ex_{Pr})$ is a marking of an event in E_{Pr}
3) $L_{Pr} = img(f_{Pr})$ is the set of labels of events in E_{Pr}
4) $f_{Pr}(e) = f(e)$ is the labelling function of events in E_{Pr}
5) $\longrightarrow \bullet_{Pr} = \longrightarrow \bullet \cap ((\longrightarrow \bullet\ E_{Pr}) \times E_{Pr})$
6) $\bullet \longrightarrow_{Pr} = \bullet \longrightarrow \cap ((\bullet \longrightarrow\ E_{Pr}) \times E_{Pr})$
7) $\longrightarrow \diamond_{Pr} = \longrightarrow \diamond \cap ((\longrightarrow \diamond\ E_{Pr}) \times E_{Pr})$
8) $\longrightarrow +_{Pr} = \longrightarrow + \cap ((\longrightarrow +\ E_{Pr}) \times E_{Pr})$
9) $\longrightarrow \%_{Pr} = \longrightarrow \% \cap ((\longrightarrow \%\ E_{Pr}) \times E_{Pr})$

Hence, $l_{Pr} \in \{\longrightarrow \bullet_{Pr}, \bullet \longrightarrow_{Pr}, \longrightarrow \diamond_{Pr}, \longrightarrow +_{Pr}, \longrightarrow \%_{Pr}\}$ are relations between events in E_{Pr}.

Fig. 1 presents the trip e-booking scenario that was initially presented in the introduction (c.f. SectionI) translated into DCR: Fig. 1 presents the DCR choreography and Fig. 2 shows the Tourist private DCR process. The choreography process is managed in the different partners processes, namely Tourist, TouristOfficer, Hotel, and CastleAdmin, to ensure a separation of concerns. *PayPass (p3)* is an internal event of the role Tourist, managed off-chain to preserve the privacy of Tourist. *PurchasePass (e1)* is an interaction sent by Tourist and received by TouristOfficer. It is managed on-chain. To execute the send event, Tourist triggers the SC from its private DCR process. TableI shows the choreography markings of Fig. 1 during a run. Each column stands for the events of the choreography. Rows indicate markings' changes as events on the left are triggered. For example, initially no event is executed nor pending and the event *e1* is included. Thus, its marking is (1,0,0). Once Tourist executes *e1*, the marking becomes (1,0,1). Partners have control over the set of internal events and interactions they are involved in. This set of events, mentionned hereinafter as a partner private DCR process, is illustrated by the Tourist's one in Fig. 2.

TABLE I
EVOLUTION OF THE MARKINGS (INCLUDED, PENDING, EXECUTED) OF THE DCR CHOREOGRAPHY PROCESS IN FIG. 1 (BEFORE CHANGES)

	Markings			
	e1	e2	e3	e4
(init)	(1,0,0)	(0,0,0)	(0,0,0)	(0,0,0)
e1	(1,0,1)	(1,1,0)	(0,0,0)	(0,0,0)
e2	(1,0,1)	(1,0,1)	(1,1,0)	(1,1,0)

Both choreography and private DCR processes in Fig. 1 and therefore in Fig. 2 are susceptible to changes. Fig. 3 presents the DCR choreography of the trip e-booking process and the Tourist private process after introducing a certain number of changes. A change is composed of a set of change elements and a combination of change operations. We define in the following these two concepts.

Definition 5. Change element Let C=(G, I, R) be a DCR choreography. Let ϵ be the set of *internal events* in G, i.e., events having one initiator $r \in R$. I is the set of *choreography interactions*. G_{Ref} is a change element (also called refinement element) iff one of the following conditions are met:

1) $G_{Ref} \in \{\epsilon \cup I \cup \longrightarrow \bullet \cup \bullet \longrightarrow \cup \longrightarrow \diamond \cup \longrightarrow + \cup \longrightarrow \%\}$
2) $G_{Ref} = (e \in \{\epsilon \cup I\}, m_e(in, pe, ex), \{\longrightarrow \bullet \cup \bullet \longrightarrow \cup \longrightarrow \diamond \cup \longrightarrow + \cup \longrightarrow \%\})$

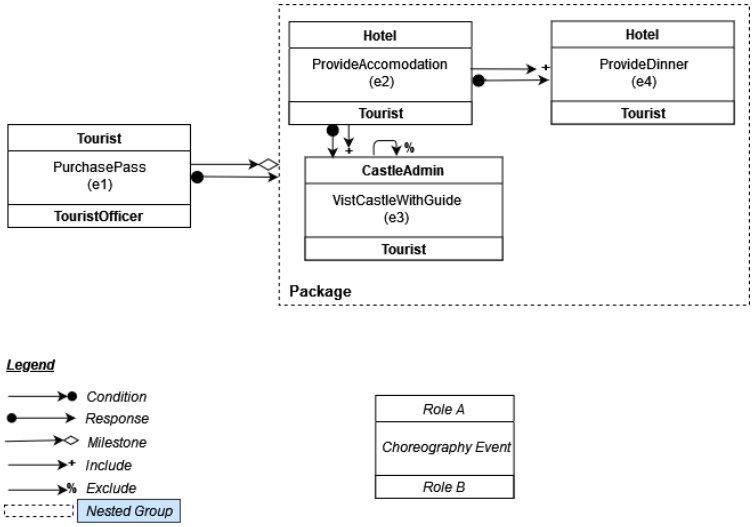


Fig. 1. DCR choreography process of the trip e-booking process

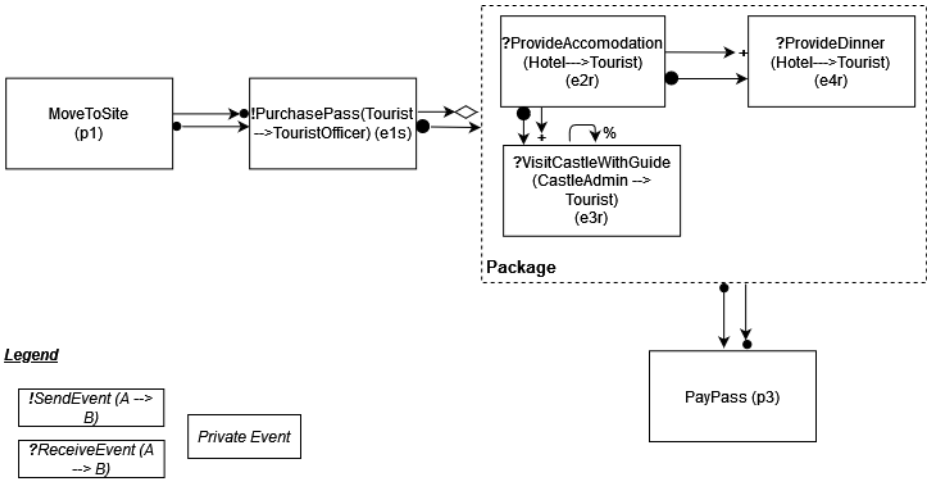


Fig. 2. The Tourist private DCR process

This means that, G_{Ref} is a refinement element if it is either (1) an atomic element, i.e., (i) an internal event in the set of internal events ϵ such as *p2* in Fig. 2, or (ii) an interaction such as *e3*, or (iii) one of the five relations such as the condition relation linking *e3* and *e2*. (2) a DCR fragment, i.e., a sub-graph with a minimal configuration: {one event, initial marking of the event, one relation}. For example in Fig. 3, change #3 consists into adding the DCR fragment ($p2$: (1,0,0), $p2 \longrightarrow \bullet e3, p2 \longrightarrow +e3, p2 \longrightarrow \%p2$).

Definition 6. Change operation To define the change operations, we refer to [9] where authors propose three change operations on DCR orchestration processes. We re-adapt these operations to be used in the context of DCR choreographies and where the change element can be one of the three types defined in **Definition 5**. Change operations are of three major types[2]:

- $C \oplus G_{Ref}$ to *ADD* the refinement element G_{Ref} to the original DCR choreography C. To apply the change, one has to compose the refinement element with the original graph, i.e., one has to take the union of events, labels, relations and markings of the two parts of the composition.
- $C \ominus G_{Ref}$ to *REMOVE* a change element G_{Ref} from C. For example, to remove an interaction, one has to remove it from the set of interactions I, its marking from the marking (In,Pe,Ex) of the graph as well as the incoming and outgoing constraints coming to/ going from this interaction.
- $C[G_{Ref} \mapsto G'_{Ref}]$ to *UPDATE* a change element. For the case of an event (internal or interaction): the *UPDATE*

[2]We use the same notation of the operations defined in [9]

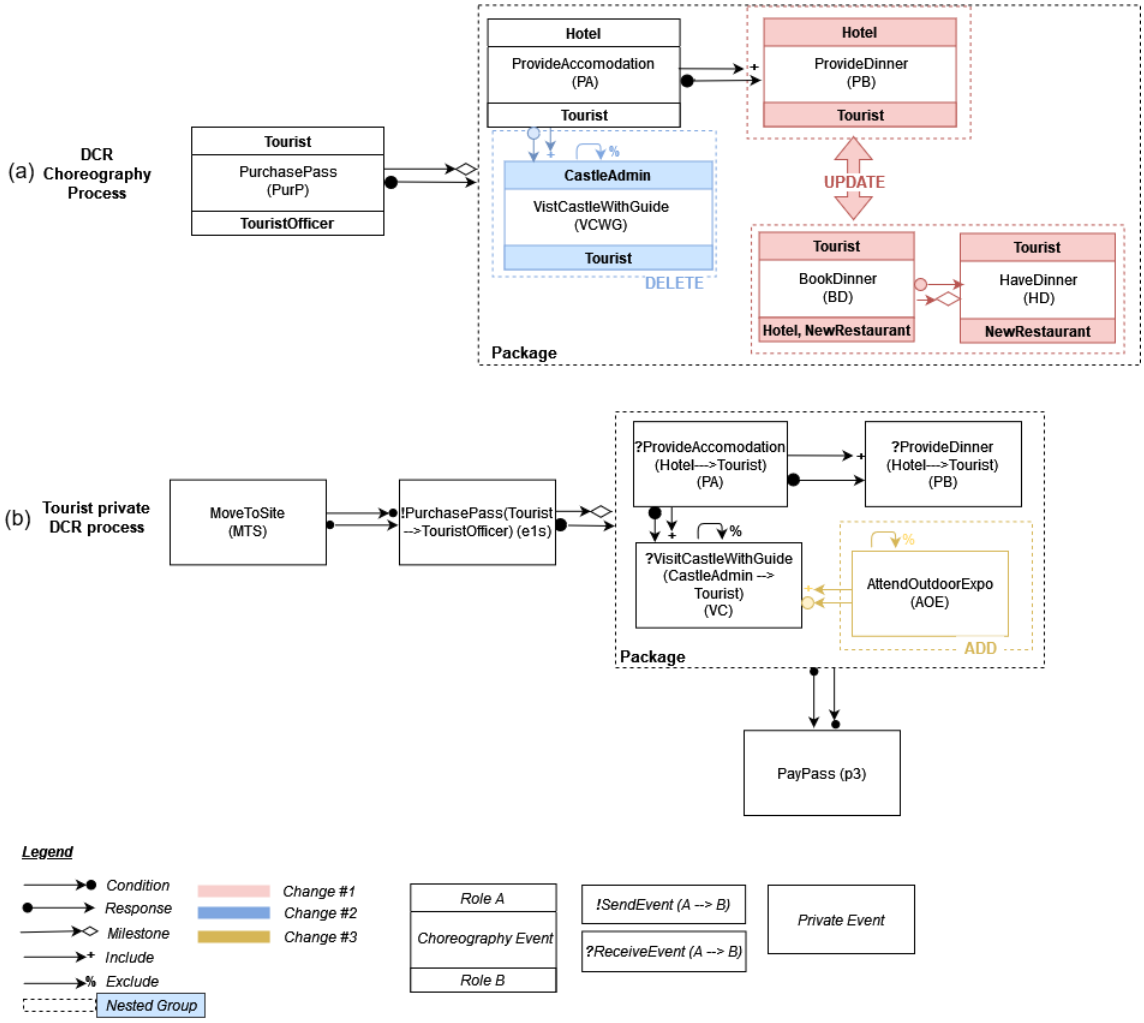


Fig. 3. DCR choreography process of the trip e-booking process after the changes

operation is used for replacing one event by another or re-labelling it. To replace, for example, one interaction with another, one has to update the set of interactions I with the new interaction, the marking of the graph and the set of incoming and outgoing constraints.

An example of an UPDATE operation is change #1 in red in Fig. 3. Here, the pass purchased by the Tourist in the event $e1$ undergoes a change: it will let the Tourist to have a dinner in NewRestaurant instead of having it in the Hotel. Hence, the TouristOfficer, who manages the pass and can add new participants, establishes a new convention with NewRestaurant. Consequently, the change operations to make are: (i) add the partner NewRestaurant, (ii) an UPDATE operation where the interaction *e4* is replaced with the DCR fragment$\{e5,e6\}$. These changes are called *public changes*.

To proceed with such a change: (i) the change should be negotiated (agreed on or not) by the involved partners, (ii) the change proposition should be examined by all involved partners, (iii) the negotiation outcome should be tamper-proof to avoid that someone diverges from the common understanding, and (iv) the change should be correctly propagated [11], [16].

III. Proposed Approach

DCR business processes monitored in the blockchain are represented into SCs as follows: the SC holds a set of activities, each assigned to an actor, and linked to an execution state. A relation matrix which summarises the execution constraints is used to update activity states based on smart-contract based execution requests.

Partners coordinate their own processes connected to the blockchain, and propose/receive changes to/from other partners (step 1 in Section. III-A). Our goal is to make it possible for each partner to (i) modify its private DCR process, and (ii) suggest a change to the DCR choreography monitored in the blockchain. If the change request is fully private, for e. g., it concerns an internal event or a relation linking two internal events, (private-to-private relation) or a relation linking an interaction to an internal event (public-to-private relation), then the private process of the partner updates accordingly. If the change is public, it is managed onchain (step 2 in

TABLE II
PROPOSED ALLOWED AND DENIED CHANGES FOR A DCR PROCESS

Type	Rule
AR1	Change condition / response / milestone relations
DR1	Inclusion of an excluded event
DR2	Exclusion of an included event
AR2	Block temporarily/ permanently an included event

Section.III-B). Public changes concern an interaction or a relation linking two interactions (public-to-public relation) or a relation linking an internal event to an interaction (private-to-public relation). Then, a negotiation stage starts (step 3 in Section. III-C), followed by a propagation stage (step 4 in Section. III-D.)

A. Step 1: Change Proposal

The role initiator defines the change of its private DCR process off-chain. She may modify its internal events and interactions, as well as relations linking events. The introduction of a change is called refinement (cf. **Definition 2**). It is done before submitting it to other partners for examination.

A set of integrity rules need to be defined to ensure the correctness-by-construction of the updated graph. A DCR graph is correct-by-construction iff it is safe, i.e., free of deadlocks and live, i.e., free of livelocks. A DCR graph is deadlock free if for any reachable marking, there is either an enabled event or no included required responses. Whereas liveness describes the ability of the DCR graph to completion by continued execution of pending response events or their exclusion. To do so, we leverage non-invasive adaptation rules, originally introduced in the context of DCR orchestrations, to DCR choreographies [11]. We divide these rules in rules describing (i) allowed change rule (AR) and (ii) denied change rule (DR) presented in TableII [3].

One can ADD/REMOVE/UPDATE condition, response and milestone relations (AR1). The only restriction is not to have cycles of condition/response relations to avoid deadlocks. However, one cannot include an already excluded event (DR1) neither can she exclude an already included event (DR2). One alternative to this is to block temporarily or permanently an event (AR2). We suppose that we want to block permanently a DCR graph G of executing an event e. We refine with the fragment Q: Q = { e: (0,1,0), g: (0,1,0)), $g \longrightarrow \bullet\ g$, $g \longrightarrow \bullet\ e$ }. Here, e can never fire (again) because it depends on g. Moreover, by excluding and including g, one can selectively enable and disable e.

In our example, the change proposal #1 consists into replacing $e4$ by the fragment composed of the events $\{e5,\ Pe6\}$. Here, one did not add an exclude relation to the already included event $e4$, i.e., (DR2) evaluates to *false*. Consequently, one is not concerned by blocking temporarily/ permanently an event, i.e., (AR2) is also verified. Only milestone and response relation are added to the graph and thus (AR1) evaluates to *true*. Moreover, change # 1 does not contain an include relation to an already excluded event and so (DR1) is also respected. Hence, the change proposal evaluates to *true* because $\forall i$, (ARi) evaluates to *true* and $\forall j$, (DRj) evaluates to *false*.

[3]The reader can check [11] for more details about denied and allowed change operations in DCR orchestrations that inspired the proposed changes.

B. Step 2: Change request for public-related changes

The SC stores the list of change requests assigned to process instances as a hashmap. Ongoing process instance changes are recorded with the identification hash of the current process instance h_{curr}. The identification hash corresponds to the IPFS hash of the process instance description[4]. This hash is generated by the change initiator upon a change request, before the SC call. During the change request lifecycle, the request is assigned to a status belonging to {*Init*, *BeingProcessed*, *Approved*, *Declined*}. Status is set to *Init* if no change request is ongoing, to *BeingProcessed* during the negotiation stage, to *Approved* or *Declined* once the change request is processed by all endorsers.

Algorithm1 presents the SC function registering a change request. The identity of the change initiator is checked: it should belong to the list of partner addresses (line 2). Then, the change request is created for the current process instance (line 3-8).The hash of the redesigned workflow is stored in h_{req} (line 4). This identification hash corresponds to the IPFS description of the requested redesigned public workflow h_{req}. The status of the change request is set to *BeingProcessed* (line 5). The addresses of the change initiator and endorsers E are attached to the request (line 6-7). Endorsing partners are, for example, in the case of adding a choreography interaction i (i) the sender and the receiver(s) of the event and , (ii) partners connected directly with the choreography interaction. The change initiator also sets two response deadlines t1 for change endorsement and t2 for change propagation to be checked by the SC (line 8-9). Finally, the SC emits a change request notification to all partners listening to the SC (line 10). If one of the change endorsers does not reply before deadline t1 during endorsement or t2 during propagation, an alarm clock triggers a SC function cancelling the change request. If one of the change endorsers does not reply before deadline t1 during endorsement or t2 during propagation, an alarm clock triggers a SC function cancelling the change request at a specified block in the future corresponding to t1 or t2. It consists into a SC function being called by incentivized users triggering the SC at the desired timestamp [18]. Upon trigger, the SC function sets the change request status to cancelled and emits an event notifying partners that the change has been cancelled. By so doing, we prevent any deadlock that could occur due to one of the partners not responding.

In Fig. 3, Change #1 is public as it concerns three partners, namely Tourist, Hotel, and NewRestaurant. Hence a negotiation must occur between the partners to reach a consensus on the proposed change before propagating it. NewRestaurant launches the change negotiation by triggering the SC. The SC updates the change requests list linked to h_{curr} with the following information: [(1) h_{req} the IPFS hash of the updated process description which comprises

[4]InterPlanetary File System (IPFS) is a peer-to-peer protocol that uses content addressing for storing and sharing files on the blockchain (https://docs.ipfs.io/).

the operation UPDATE(e4) with (e5+e6), (2) the list of endorsers: $\{address_{Hotel}, address_{Tourist}\}$, (3) Change negotiation deadline t1 = 72h, (4) Change propagation deadline t2 = 120h]

Algorithm 1: Request change smart contract function

Data: $changeRequests$ the list of change requests, E the list of endorser addresses, h_{curr} the current ipfs workflow hash, h_{req} the ipfs hash of requested change description, $t1$ the deadline timestamp for change endorsement, and $t2$ the deadline timestamp for change propagation
Result: emits change request notifications to endorsers
Function `requestChange` (h_{curr}, h_{req},$E, t1, t2$) **:**
  require $msg.sender$ belongs to the list of business partners;
  if *changeRequests[h_{curr}].status* $== Init$ **then**
    set $changeRequests$[h_{curr}].$h_{req} \leftarrow h_{req}$;
    set $changeRequests$[h_{curr}].status $\leftarrow$ "$BeingProcessed$";
    set $changeRequests$[h_{curr}].initiator $\leftarrow msg.sender$;
    set $changeRequests$[h_{curr}].endorsers $\leftarrow E$;
    set $changeRequests$[h_{curr}].t1 $\leftarrow t1$;
    set $changeRequests$[h_{curr}].t2 $\leftarrow t2$;
    emit RequestChange(h_{curr}, h_{req}, E, $msg.sender$);
  end
  else
    emit Error; `// an ongoing change request is being processed`
  end
End Function

C. Step 3: Change negotiation for public-related changes

All partners subscribe to the change request events emitted by the SC. Endorsing partners must send their decision request to the SC based on the rules in Table. II. If the change once computed on the endorser's process respects all ARi and DRj rules, then the endorser approves the request. It is otherwise rejected. The rules checks are manual and can be automated in the future work. The SC collects the different decisions from the endorsers to lock (or not) the choreography instance and proceed (or not) with the change. We detail both stages hereinafter.

Algorithm 2 presents the SC function receiving one endorser's decision. the alarm clock should not have been raised (line 2), the endorser address e_s should belong to the list of registered addresses (line 3), and not having answered to the change request already (line 4). The change request should also be processable, i.e., its status should be set to *BeingProcessed* (line 5). If all conditions are met, the endorser response rsp is processed. If rsp equals 1 (line 6), the endorser has accepted the change. Its response is saved into the change endorsement list (line 7), the notification of acceptance is sent to all endorsers as well as the change initiator (line 8), and

Algorithm 2: Endorser decision management smart contract function

Data: $changeRequests$ the list of change requests, e_s the endorser address, E the list of registered endorsers, h_{curr} the hash of the current workflow, h_{req} the hash of the desired workflow, rsp the endorser response $\in \{0, 1\}$
Function `endorserRSP` (h_{curr}, e_s, rsp) **:**
  require($block.timestamp <= changeRequests[h_{curr}].t1$);
  require($e_s \in E$);
  require($changeRequests[h_{curr}].cEdr[e_s]$!= 1);
  require($changeRequests[h_{curr}].status ==$ ("$BeingProcessed$");
  if $rsp == 1$ **then**
    set $changeRequests[h_{curr}].cEdr[e_s] \longleftarrow 1$;
    emit AcceptChange(h_{req}, e_s);
    lockInstanceChecker(h_{curr})
  end
  else if $rsp == 0$ **then**
    `// declineapprovalOutcomes`
    set $changeRequests$[h_{curr}].status $\leftarrow$ "$Declined$";
    emit DeclineChange(h_{req}, e_s);
  end
  else
    emit Error(h_{req}, e_s);
  end
End Function

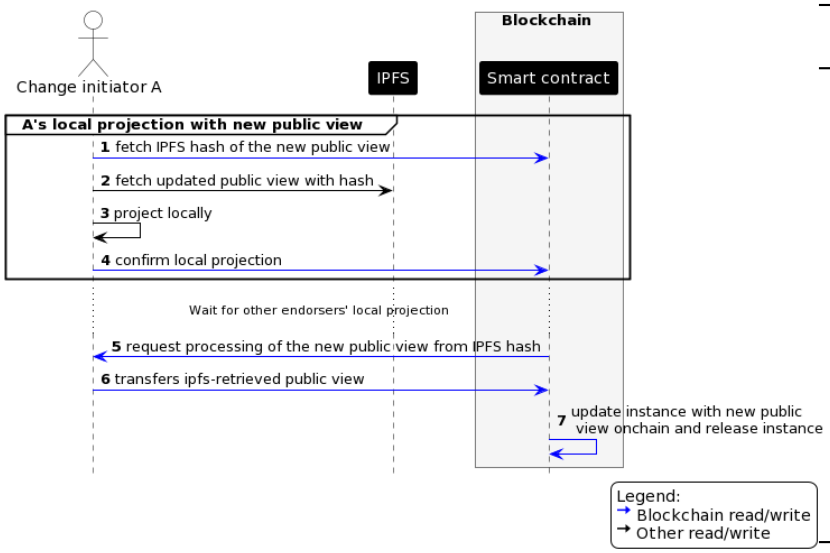
the SC checks whether the instance needs to be locked (line 9). The $lockInstanceChecker$ function assesses whether all endorsers have accepted the change: the $cEdr$ list should be filled with ones. At this stage, no further execution of included events is allowed and the mechanism waits for pending events to terminate. The change status is then updated to *Approved*.

In our example, we suppose that both endorsers confirmed the change request ($rsp_{Hotel} = 1$ and $rsp_{Tourist} = 1$) while respecting t1. The SC locks the instance for change propagation. As it manages the negotiation process, a tamper-proof record of the negotiation is accessible by all partners. This prevents conflicts and eases potential claim resolutions.

D. Step 4: Change propagation

Change propagation is to apply the change effect after the negotiation phase succeeds to (i) the affected partners DCR public processes, (ii) each partner propagates the change effect to its private DCR process.

Indeed, Fig. 4 depicts the sequence diagram of the change propagation interactions taking place between partners and the SC. Each partner projects locally the DCR choreography in its projection using the process description given in the IPFS hash (Fig. 4 step 1-4). Participant A first fetches the IPFS hash of the new choreography process stored on-chain (step 1), and uses the hash to retrieve the description stored in IPFS (step 2). Using this information, A projects locally the new version of the process, merging its private process with the



Algorithm 3: Confirm change propagation smart contract function

Data: $changeRequests$ the list of change requests, e_s the sender address, E the list of registered endorsers, h_{curr} the hash of the current workflow

Result: manages the record of projections of the new public view

Function `confirmProjection(`h_{curr}, e_s`)`**:**

require($e_s \in E$);

require($block.timestamp <= changeRequests[h_{curr}].t2$);

require(changeRequests[h_{curr}].didPropagate[id] != 1);

set changeRequests[h_{curr}].didPropagate[id] $\longleftarrow$ 1;

emit LogWorkflowProjection(h_{curr});

End Function

Fig. 4. Sequence diagram of the propagation stage illustrating the interactions between partners and the SC with A being the change initiator

updated public activities. Once completed, A notifies the SC to confirm the projection (step 4). Algorithm 3 presents the function triggered by partners to confirm the projection to the SC. A list $didPropagate$ keeps track of the propagation status, i.e., it records the private projection of each partner. The function checks that the partner belongs to the list of endorsers (line 2), that the alarm clock has not been raised (line 3), and that the endorser has not projected locally yet (line 4). Each participant must proceed before the change propagation deadline $t2$. Else, the propagation is cancelled, and the instance returns to its initial state before the change request. Other endorsers follow the same steps.

The SC detects all local projections once $didPropagate$ is filled with ones and notifies the change initiator. The change initiator then retrieves the new DCR choreography that was saved into IPFS using h_{req} (Fig. 4 step 5) and forwards it to the SC (Fig. 4 step 6). The SC updates the relations and markings stored into the process instance and resets the change status of the workflow instance: a new change request can be processed (Fig. 4 step 7). In total, all participants must complete two transactions with the SC and one transaction with IPFS. The change initiator must also complete two additional transactions with the SC to update the DCR choreography instance definition stored on-chain and unlock the instance.

In our motivating example, the propagation of change #1 occurs with all endorsers Tourist, Hotel and Restaurant updating their private DCR process with the approved change. To do so, they retrieve the change description stored in IPFS under h_{req}. They project the updated public change description on their role. For example, Tourist will retrieve the events $\{e5, e6\}$. Tourist then combines this projection with its private events $\{p1,p2,p3\}$. Once all projections have been done and notified to the SC, the change initiator NewRestaurant finally triggers the SC to update the DCR choreography of the running instance with the updated process description e.g., the updated relation matrices, event markings and access controls.

In this section, we give the different steps in order to migrate the running instance of the DCR choreography process to the new choreography description. We precise that, unless a new change request is triggered, for all new instances to arrive, they will be handled according to the most new description of the choreography stored in the SC.

IV. Proof of Change Propagation Correctness

In this section, we give the properties which are necessary and sufficient to guarantee the correctness of the change propagation. In Section. IV-A, we demonstrate structural and behavioral compatibility between partners public processes. In Section IV-B, we illustrate how to guarantee consistency between one partner's private and public process.

A. Criterion 1: Compatibility between partners public processes

a) Structural compatibility: Structural compatibility checks consist of ensuring that there is at least one potential send message assigned to a partner with a corresponding receive message assigned to another partner.

Structural compatibility is ensured by the notion of *projection* applied when generating the public processes of the partners on-chain (cf. Section III-D). To explain more, we mentioned that the choreography process is managed on-chain with a unique IPFS hash stored in the SC. When generating her public DCR process, each participant, identified by a public blockchain address, fetches this IPFS hash to retrieve the description of the choreography process. Then, she projects this description over her role (noting that we refer to participant or role as the same thing). In other words, if she appears in the sender role of the choreography event, then she generates a send event. Otherwise, i.e., if she is in the list of the receivers roles, she thus generates a receive event. The SC records the behavior of the different choreography partners/roles via the function presented in Algorithm 3 in Section III-D. In this way, one can assure that all partners have

projected correctly and trustfully the choreography process over their roles and thus that public processes are generated correctly, to each send message assigned to a partner, there is a potential receive message assigned to another partner.

b) Bahavioral compatibility: In Table II, p. 10, we give expressive constructs describing allowed and denied changes to be applied to a DCR graph. As we mentioned previously in Section III-A, the application of the mentioned constructs leads to a correct-by-construction DCR graph which is *safe* and *live*. Moreover, the changes derived from these rules are *complete*, which means any change to apply to a DCR graph is equivalent to the application of a single rule or a combination of the aforementioned rules (proof of the completeness of the changes can be found in [11]).

By relying on these rules and to ensure the compatibility between public processes, we introduce the following property where c is a change, r, E are respectively c initiator and endorsers, G_r,$G_{r'}$ are respectively the public DCR process of r and r' where $r' \in$E$\backslash r$, $effect(c)_{\|G_r}$ the effect of a change c over r and $effect(c)_{\|G_r}$: $\Rightarrow G_r$ refers to computing the new state of the graph G_r after applying the effect of c over r, then the proposed property stands as follows:

Property 1. *if* $effect(c)_{\|G_r}$: $\Rightarrow$ G_r is correct-by-construction and $\forall r' \in E$, $effect(c)_{\|G_{r'}}$: $\Rightarrow G_{r'}$ is correct-by-construction *then* G_r and $G_{r'}$ are compatible $\forall r \in E$, $\forall r' \in E \setminus r$.

Proof. Property 1. states that if G_r is correct-by-construction and if $\forall r' \in E$, $G_{r'}$ are also correct-by-construction, then compatibility is verified. In fact, a public DCR process G_r is correct-by-construction means that computing the effect of a change c over r introduces no deadlocks in G_r (see Section. III-A). Thus, if the DCR public models of the change initiator and endorsers are safe, i.e., no deadlocks can occur, then they are able to communicate in a proper way after the change and are consequently compatible with each other. This is enforced by the SC as it maintains the tamper-proof record for the endorsement and application of the change effect across partners. Another correctness criterion is checking the consistency between one partner's private and public DCR processes.

B. Criterion 2: Consistency between partners public and private processes

In this section, we demonstrate how to check that one's partner private process is consistent with her public process after the change.

We give precisely the different properties which are necessary and sufficient to construct one partners private and public processes which are behaviourally equivalent. Consequently, if the effect of one (or many) change(s) over one partner's public process leads to a live and safe process, then applying the effect of this change on the partner's private process will lead to a safe and live private process as well.

We are inspired from the work done by [19]. In [19], authors base their particular correctness criteria on graph equivalence by checking compliance under inheritance relations. They state that a subclass inherits the behavior of the corresponding superclass if and only if the subclass is constructed from the superclass by applying one or many of the transfer rules. The transfer rules represent the types of changes typically used to adapt a workflow process definition by adding/ removing cyclic structures, sequences, parallel and alternative branches. In that case, authors demonstrate that one cannot distinguish the behavior of two workflow nets belonging respectively to the super and subclass.

Here, we go through the same analogy and define the private DCR process P_r of a role r as a subclass of the public DCR process G_r. But before that we give the definition of the abstraction operator used to hide tasks in a DCR graph as the following:

Definition 7. Abstraction operator Let $G = (E, M, L_0, f_0, \longrightarrow \bullet, \bullet \longrightarrow, \longrightarrow \diamond, \longrightarrow +, \longrightarrow \%)$ be a DCR process and ϵ the set of internal events in G. For any $e \in \epsilon$, the abstraction operator τ_ϵ is a function that renames all labels of events in ϵ to the silent event τ. Formally, $\tau_\epsilon(G)= (E, M, L_1, f_1, \longrightarrow \bullet, \bullet \longrightarrow, \longrightarrow \diamond, \longrightarrow +, \longrightarrow \%)$ such that $\forall e \in \epsilon$, $f_1(e) = \tau$, $\forall e' \in E \setminus \epsilon$, $f_1(e') = f_0(e')$ and $L_1 = img(f_1)$.

Definition 7. describes the abstraction operator permitting to hide some events in a DCR graph. To hide one event is equivalent to rename it to a silent action. Silent actions are denoted with the label τ, i.e., only events labels different from τ are observable.

This will lead us to **Property 2.** which states that the private DCR process P_r of a role r is a subclass of the public DCR process G_r as follows:

Property 2. $\forall r \in E$, $P_r = G_r + N$ where N is the new behavior. P_r is a subclass of the super class G_r and $G_r = P_r + \tau_\epsilon(N)$, i.e., P_r and G_r are behaviorally equivalent.

Proof. Property 2. states that for every role r in a DCR choreography process, the private DCR process P_r of r inherits the behavior of G_r and that by hiding the new behavior N in P_r, one cannot distinguish the behavior of P_r and G_r. The new behavior N is simply the set of internal DCR fragments in P_r. Internal DCR fragments are added between sequential choreography tasks as one can see in the running example in Fig. 2 in Section. II. In fact, as we previously precise in **Definition 4.**, p 9, P_r comprises the public interactions in addition to the internal events related to this partner r. We noted ϵ_r the set of internal events related to one role r. ϵ_r describes the behavior of role r that is not visible to other partners.

To summarize, the transfer rule we apply in order to construct a private DCR process P_r of one role r out from the public DCR process G_r of r is by adding internal DCR fragments between sequential choreography tasks in G_r. Thus, following this rule, one can say that P_r is a subclass of G_r. Consequently, if the effect of one (or many) change(s) over one partner's public process leads to a live and safe process, then applying the effect of this change over the partner's private process will lead to a safe and live private process as well. In other words, one partner's private process is consistent with her public process after the change.

V. Implementation and evaluation

A. Implementation

In our previous work [20], we presented a solution aiming at executing DCR choreographies in a hybrid on/off-chain fashion. The DCR choreography description comprises events descriptions (i.e., labels, roles, markings), relation matrices, and actors linked to events. Here, we leverage this platform to integrate change at the process instance level [5].

We use a Ganache testnet to deploy a public SC S which manages each process. S comprises (1) execution constraint rules, (2) a list of workflows initially empty, and (3) the list of change requests linked to the list of workflows. At the time of writing, 1ETH=2581,86 $. The initial cost of deployment of S is 0.10667554 ETH (439.21$) for a gas usage of 3,555,855. Additionally, a SC manages roles authentication and access control rules. Its deployment takes 1,953,149 gas (0.05859442 ETH or 241.25$). The SC is deployed in Ropsten at: 0x523939C53843AD3A0284a20569D0CDf600bF811b[6]. For each workflow, RoleAdmin (1) generates the DCR choreography bitvector representation, (2) saves the textual DCR choreography input to IPFS, and (3) registers the new workflow on-chain. The workflow is identified by the IPFS unique hash.

Each partner can edit the running instance. Editing is done using the panel manager, a tool to update DCR graph descriptions. Users can add private and choreography interactions, as well as condition, response, include, exclude, and milestone relations. They can also use the panel manager to remove and update events and relations. The panel manager implements integrity rules presented in subsection III-A: the panel verifies the soundness of a desired change operation. Hence, we obtain a redesigned DCR graph that is correct-by-construction. After edition, the panel manager triggers the SC if it detects a public change. The SC registers the request and forwards it to the identified partners. Each partner accesses the change request and answers back to the SC. If the change request is accepted by all, change propagation starts.

B. SC evaluation costs

The initial cost for deploying the motivating example instance is 0,00933308 ETH (24,097$) for a gas usage of 311,103. Indeed, the consensus algorithm used in the Ethereum blockchain is a proof of work [3], hence each SC transaction excepting read transactions are payable to compensate miners from computation costs. We evaluate the transaction costs to assess the computation costs related to the change negotiation and propagation functionalities.

In our motivating example, three changes occur. Change#1, initiated by NewRestaurant, is fully public: *e5* and *e6* replace *e4*, and two public-to-public relations (response and milestone) are added. In the following, we investigate the public negotiation and propagation SC costs for this change.

[5]Code of the implemented prototype augmented with change management is accessible at https://anonymous.4open.science/r/adaptiveDCRs-2CC3/

[6]This address can be used with Etherscan to access the record of transactions.

TableIII presents the gas usage induced by the execution of the SC during the negotiation and propagation stages. It is used to compensate miners for their computation power in the blockchain cryptocurrency. The table also presents the transaction costs in ETH and USD.

Regarding the negotiation stage, Tourist first launches the change request for the replacement of one public task by a new fragment of two public tasks. The transaction fees for the request are 0,00639582 ETH, and are the highest fees of the negotiation stage. Indeed, the fee to be paid to decline or accept a role is worth around 0.0015 ETH. Nonetheless, all fees are of the same order of magnitude (0.001 ETH). *Regarding the propagation stage*, the transaction fees of the SC correspond to two stages. First, the change endorsers apply the change effect to their private processes. No transaction fee is requested to fetch the IPFS hash of the new DCR choreography. However, a transaction fee is necessary to update the SC list *didPropagate* recording the projections. The SC notification of the local update is worth 0,00201296 ETH and 0,0020115 ETH for both endorsers (around 5$ per local projection). The change initiator finally updates its projection. The cost to switch the workflow locally is 0,00175296 ETH. NewRestaurant sends a transaction to update the DCR choreography on-chain using the same tool used to deploy a new instance on-chain. The cost for switching the DCR choreography on-chain is 0,02642992 ETH. It is one order of magnitude higher compared to other transaction fees, but close to the cost of instantiating a new instance on-chain, due to the update of relation matrices and markings. Hence, propagation transaction fees are higher than the negotiation ones. Additionally, the cost of the propagation mainly comprises the cost of the DCR choreography update. *Execution times*, represent the results obtained after the enactment of one trace. The reported execution time factors the transaction confirmation time obtained on the test network. In average, the execution time of on-chain interactions is 14.8s. Additionally, the average time for IPFS transactions is 7.6ms. The change initiator NewRestaurant needs to process four on-chain transactions and two off-chain transactions with IPFS. Both endorsers TouristOfficer and Tourist must process three on-chain transactions and two IPFS transactions. Hence, in total, the whole cycle of change management takes 152.6s if all participants launch their transactions on trigger (4+3+3 on-chain transactions requiring 14.8s in average, and 2+2+2 IPFS transactions requiring 7.6ms in average).

VI. Related Work

In the following section, we present the related work regarding change mechanisms in cross-organizational business processes as well as approaches to prove dynamic change correctness in business processes.

a) Change mechanisms in cross-organizational business processes: Change management at runtime in procedural processes has been studied in [7] where change propagation algorithms ensure behavioral and structural soundness of choreography partners private processes after the change. In [21], authors consider the change negotiation phase but

TABLE III
SC CHANGE PROPAGATION GAS COSTS AND GAS FEES

Stage	Step	Partner	Gas	Cost(ETH)	Cost($)
Nego.	LaunchNego	NewRestaurant	213194	0,00639582	16,513
	Case Decline	TouristOfficer	46773	0,00140318	3,623
	Case Accept	TouristOfficer	78999	0,00157998	4,079
		Tourist	86428	0,00181256	4,68
Propag.	Upd. projection	TouristOfficer	96448	0,00201296	5,197
	Upd. projection	Tourist	96375	0,0020115	5,193
	Upd. projection	NewRestaurant	87648	0,00175296	4,526
	Upd. SC instance	NewRestaurant	1321496	0,02642992	68,238

no mechanism is proposed to ensure that all partners have trustfully applied the change, and no blockchain is used to deal with this problem. In addition, authors do not consider changes on running instances nor public-to-private change propagation.

Change management has also been studied in DCR processes, mainly through runtime changes. The first efforts appear with the notion of DCR fragments where simple change/add/remove operations are implemented [9]. Authors follow the *build-and-verify* approach to apply incremental changes to the fragments. This approach consists of the continuous iterations of (i) modeling, (ii) deadlocks and livelocks freedom verification, and (iii) executing until a further adaptation is required. Nonetheless, partner trust into change propagation of DCR choreographies is not addressed in this work. In [22], authors use a *correct-by-construction* approach on running instances of DCR graphs. Starting from a user-defined change, authors define a reconfiguration workflow. During the transition period, old requirements are disabled and verified subpaths of activity executions are enabled. This setting holds until new requirements are verified. However, not every reconfiguration problem has a solution and for every change, one has to build a new reconfiguration workflow. It requires heavy calculations to discover the verified subpaths, which is not easy for large models. Finally, in [11], authors use a set of rules ensuring the correctness of new instances of DCR graphs by design. New change operations must respect these rules to prevent a misbehavior.

Regarding change management in blockchain-enabled processes, in [5], authors propose an approach that allows collaborative decisions about (1) late binding and un-binding of actors to roles in blockchain-based collaborative processes, (2) late binding of subprocesses, and (3) choosing a path after a complex gateway. A policy language enables the description of policy enforcement rules such as who can be a change initiator and who can endorse a change. However, authors do not consider ADD/REMOVE/UPDATE change operations like we do. Additionally, the private processes of roles are not considered and neither is the propagation of the effect of the new decisions over partners.

To summarize, most related work consider change in process orchestrations only [9], [11]. Additionally, approaches binding actors to roles in a process collaboration [5] currently push the burden of checking the transitive effect of new changes onto the new parties. This checking, likely done in a manual way, which can lead to errors. Finally, even when the change propagation is dealt with, the proposed approach does not provide neither a mechanism that ensures choreography partners project the change and propagate it trustfully, nor proofs of change propagation correctness.

b) Correctness criteria for dynamic changes: In [19], authors said that in a running business process under a change, it is not always possible to transfer an instance nor is it acceptable to shut down the system, transfer all instances, and restart using the new procedure. Thus, it is primordial to set down some criteria that guarantee correctness while applying the change and permits to avoid dynamic change bugs such as duplication of work, skipping of tasks, deadlocks and livelocks. In this manner, instances can be migrated to the changed schema of the process without causing such problems.

Approaches either found their correctness criteria (called also soundness criteria) on *graph equivalence* or *trace equivalence*. Graph equivalence is to compare the process schema before and after the change or to map the process instance graph to the changed schema [19]. In this context, approaches vary from those that check compliance under inheritance relations of an instance on the old schema with the new schema. We used this criterion in our approach and give details in Section. IV-B. Other approaches are based meanwhile on valid mapping where each completed activity of the running instance is also contained in the new schema and all control and data dependencies in this instance have counterparts in the new schema.

Trace equivalence is based on the execution history of the running instance. Approaches either (i) use complete history information by replaying the execution history of the running instance on the changed schema or (ii) depict the aggregated migration conditions of instances especially in case of large number of instances.

More details can be found in [23] where authors give a complete survey on correctness criteria for dynamic changes.

VII. DISCUSSION AND CONCLUSION

In this paper, we propose a change propagation mechanism to bring adaptiveness to trustworthy execution of declarative choreography instances. Our approach comprises three main steps. First, the change introduction, where a partner in a running DCR choreography instance wants to change its private process. Here, we declare rules that specify the allowed and prohibited changes. These rules provide a correct-by-construction DCR choreography after the change and ensure that no deadlocks nor livelocks occur. All changes are applied at the process instance level. Local changes are managed off-chain. Changes impacting an interaction start on on-chain

negotiation phase. If the negotiation succeeds, the change effect is propagated to the partners affected directly by the change. We suppose that all partners project trustfully the updated DCR choreography. For all new instances to arrive at the system, they will be handled according to the most new description of the choreography stored in the SC.

Here, SC transactions act like "approval check points" during change negotiation and propagation. Hence, claim resolution is eased between partners in case of a misbehavior as the blockchain stores the negotiation and propagation history on-chain. For example, when a partner wrongfully projects the change and creates a behavioral incompatibility after the change, execution logs can be used to check who is the source of and what is the erroneous behavior.

We present a prototype implementation as a proof-of-concept to evaluate the technical feasibility of the approach, and evaluate it experimentally by looking at transaction fees for a typical change. We leverage IPFS temporarily during the negotiation and propagation stages to process the updated DCR choreography for cost optimization considerations. Only a hash of the DCR process is stored into the SC. Our experiments show that the transaction fees required for the propagation are one order of magnitude higher than the ones for the negotiation, as the cost of the propagation mainly comprises the cost of the DCR choreography update. We appraise these costs to have more significance for heavy negotiation scenarios, and to be proportional to the number of participants involved in a public change, as the more participants, the more interactions are necessary with the SC.

In this paper, we consider two criteria for change propagation correctness. The first one is to check compatibility between public DCR processes of partners. We distinguish between structural and behavioral compatibility. This ensures that the DCR choreography is safe and terminates in acceptable state, i.e., is deadlock free after the change. We prove as well consistency between one partner's private and public DCR processes. Consequently, if the effect of one (or many) change(s) over one partner's public process leads to a live and safe process, then applying the effect of this change on the partner's private process will lead to a safe and live private process as well.

A limitation of the approach is the fact that the change initiator specifies the endorsers. This can be handled differently by considering a pre-specified list of the endorsers and the choreography participants agree on this list before starting the process instance [5]. In this way, the agreement on change negotiation and propagation can be placed off-chain. An on-chain transaction saying that the agreement is reached is stored in a multi-signed document in IPFS (this might require the use of a different blockchain platform). However, even with multi-sig mechanisms, the risk of private key loss remains and recovery schemes such as using secured wallets should be investigated [24]. Finally, governance should also be considered carefully when choosing the access control setup. For public blockchains, not every endorser should necessarily run their own full node to preserve the consensus. For permissionned blockchains, governance should be well shared between change endorsers to avoid any tampering or transaction misuse.

REFERENCES

[1] J. Mendling, I. Weber, W. V. D. Aalst *et al.*, "Blockchains for business process management-challenges and opportunities," *ACM TMIS*, 2018.

[2] I. Weber, X. Xu, R. Riveret, G. Governatori, A. Ponomarev, and J. Mendling, "Untrusted business process monitoring and execution using blockchain," in *Business Process Management: 14th International Conference, BPM 2016, September*. Springer, 2016, pp. 329–347.

[3] V. Buterin *et al.*, "A next-generation smart contract and decentralized application platform," *white paper*, vol. 3, no. 37, 2014.

[4] M. F. Madsen et al., "Collaboration among adversaries: distributed workflow execution on a blockchain," in *FAB*, 2018, p. 8.

[5] O. López-Pintado, M. Dumas, L. García-Bañuelos, and I. Weber, "Controlled flexibility in blockchain-based collaborative business processes," *Information Systems*, vol. 104, p. 101622, 2022.

[6] X. Xu, I. Weber, and M. Staples, "Blockchain patterns," in *Architecture for Blockchain Applications*. Springer, 2019, pp. 113–148.

[7] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert, "Dealing with change in process choreographies: Design and implementation of propagation algorithms," *Information systems*, vol. 49, pp. 1–24, 2015.

[8] W. Van Der Aalst et al., "From public views to private views–correctness-by-design for services," in *Web Services and Formal Methods: 4th International Workshop, WS-FM 2007, Brisbane, Australia, September 28-29, 2007. Proceedings 4*. Springer, 2008, pp. 139–153.

[9] R. R. Mukkamala, T. Hildebrandt, and T. Slaats, "Towards trustworthy adaptive case management with dynamic condition response graphs," in *EDOC*. IEEE, 2013, pp. 127–136.

[10] S. H. Ryu, F. Casati, H. Skogsrud, B. Benatallah, and R. Saint-Paul, "Supporting the dynamic evolution of web service protocols in soa," *TWEB*, 2008.

[11] S. Debois, T. T. Hildebrandt, and T. Slaats, "Replication, refinement & reachability: complexity in dynamic condition-response graphs," *Acta Informatica*, vol. 55, no. 6, pp. 489–520, 2018.

[12] W. M. van der Aalst and M. Weske, "The p2p approach to interorganizational workflows," in *Advanced Information Systems Engineering: 13th International Conference, CAiSE 2001 Interlaken, Switzerland, June 4–8, 2001 Proceedings 13*. Springer, 2001, pp. 140–156.

[13] A. Khebizi, H. Seridi-Bouchelaghem, B. Benatallah, and F. Toumani, "A declarative language to support dynamic evolution of web service business protocols," *Service Oriented Computing and Applications*, vol. 11, no. 2, pp. 163–181, 2017.

[14] S. Debois, T. Hildebrandt *et al.*, "Safety, liveness and runtime refinement for modular process-aware is with dynamic sub processes," in *FM*, 2015.

[15] G. Decker and M. Weske, "Behavioral consistency for b2b process integration," in *CAISE*. Springer, 2007, pp. 81–95.

[16] T. T. Hildebrandt et al., "Declarative choreographies and liveness," in *FORTE*. Springer, 2019, pp. 129–147.

[17] A. Brahem, T. Henry, S. Bhiri, T. Devogele, N. Laga, N. Messai, Y. Sam, W. Gaaloul, and B. Benatallah, "A trustworthy decentralized change propagation mechanism for declarative choreographies," in *International Conference on Business Process Management*. Springer, 2022, pp. 418–435.

[18] C. Li and B. Palanisamy, "Decentralized privacy-preserving timed execution in blockchain-based smart contract platforms," in *HiPC*. IEEE, 2018, pp. 265–274.

[19] W. M. Van Der Aalst and T. Basten, "Inheritance of workflows: an approach to tackling problems related to change," *Theoretical computer science*, vol. 270, no. 1-2, pp. 125–203, 2002.

[20] T. Henry, A. Brahem, N. Laga, J. Hatin, W. Gaaloul, and B. Benatallah, "Trustworthy cross-organizational collaborations with hybrid on/off-chain declarative choreographies," in *Service-Oriented Computing: 19th International Conference, ICSOC 2021, Virtual Event, November 22–25, 2021, Proceedings 19*. Springer, 2021, pp. 81–96.

[21] C. Indiono and S. Rinderle-Ma, "Dynamic change propagation for process choreography instances," in *OTM Conferences*. Springer, 2017, pp. 334–352.

[22] L. Nahabedian, V. Braberman, N. D'ippolito, J. Kramer, and S. Uchitel, "Dynamic reconfiguration of business processes," in *BPM*. Springer, 2019, pp. 35–51.

[23] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems—-a survey," *Data & knowledge engineering*, vol. 50, no. 1, pp. 9–34, 2004.

[24] F. Xiong, R. Xiao, W. Ren, R. Zheng, and J. Jiang, "A key protection scheme based on secret sharing for blockchain-based construction supply chain system," *IEEE Access*, 2019.