A trustworthy decentralized change propagation mechanism for declarative choreographies

Amina Brahem, Tiphaine Henry, Sami Bhiri, Thomas Devogele, Nassim Laga, Nizar Messai, Yacine Sam, Walid Gaaloul, Boualem Benatallah

▶ To cite this version:

Amina Brahem, Tiphaine Henry, Sami Bhiri, Thomas Devogele, Nassim Laga, et al.. A trustworthy decentralized change propagation mechanism for declarative choreographies. 20th International Conference on Business Process Management, Sep 2022, Munster, Germany. pp.418-435, 10.1007/978-3-031-16103-2_27 . hal-04295596

HAL Id: hal-04295596
https://univ-lyon3.hal.science/hal-04295596v1

Submitted on 20 Nov 2023

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

A trustworthy decentralized change propagation mechanism for declarative choreographies

Amina Brahem[1,3], Tiphaine Henry[2,4], Sami Bhiri[3], Thomas Devogele[1], Nassim Laga[2], Nizar Messai[1], Yacine Sam[1], Walid Gaaloul[4], and Boualem Benatallah[5]

[1] LIFAT, University of Tours, Tours, France, amina.brahem@univ-tours.fr
[2] Orange Labs, Paris, France, tiphaine.henry@orange.com
[3] OASIS, University Tunis El Manar, Tunis, Tunisia
[4] Telecom SudParis, UMR 5157 Samovar, Institut Polytechnique de Paris, France
[5] Dublin City University, Ireland

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. Thus, the change effect must be propagated to partners of the choreography affected by the change. In this paper, we propose a new 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 DCR process. A change impacting other partners is propagated to their affected processes using a SC. The change propagation mechanism ensures the compatibility checks between public DCR processes of the partners. We demonstrate the approach's feasibility through an implemented prototype.

Keywords: Process choreography, Change propagation, DCR graph, SC

1 Introduction

Blockchain technologies have emerged to serve as a trust basis for the monitoring and execution of business processes [18], 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 SCs and its ability to manage decentralized, peer-to-peer interactions [10].

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) graphs [3]. However, dealing with changes in blockchain-enabled business processes remains an open research issue [17].

Business processes managed by "static" SCs cannot be upgraded because the SCs are immutable once deployed. Efforts exist to support versioning in SCs [22]. 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 [22]. 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, 9]. In a running choreography instance, a change may consist of a simple change operation (ADD/REMOVE/UPDATE) or combination of change operations [4, 7]. 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, 20]. 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.

One has also to ensure that neither the structural nor behavioral compatibility of partners processes are violated after a change [1, 7, 11, 12]. 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[12]. 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 change [7, 11]. 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.

To the best of our knowledge, the integration of change management, and especially change propagation, in blockchain-based declarative choreographies management systems has not been studied. In this paper we focus on the fol-

lowing research question: *(RQ) How to guarantee correct change propagation in declarative blockchain-based DCR choreography process instances?*

We propose a change mechanism to bring adaptiveness to the trustworthy execution of declarative choreographies. We adopt the three levels of granularity: (i) choreography, (ii) public and (iii) private processes used in imperative languages such as BPMN and apply it to the declarative language called DCR graphs [7]. With DCR, processes are modelled as a set of events linked together with relations (a kind of temporal dependencies) [5, 6]. In [14], authors propose an approach for a trustworthy deployment and execution of DCR choreographies. Choreography participants build incrementally the choreography process managed by a SC. Meanwhile, participants execute their private events off-chain in their local process execution engine. We build on and extend this work with the change management mechanism, focusing on the introduction, negotiation and propagation phases at the process instance level. Similarly to declarative languages, a DCR graph is specified as a set of rules. These rules are interpreted at runtime. As they represent business requirements, it is easier to add or update constraints if a requirement changes [5].

Our approach allows a partner in a running DCR choreography instance to change its private process. Changes affecting private activities are applied off-chain while changes impacting interactions with other partners are managed on-chain through the SC [14]. Changes are mainly ADD/ REMOVE and UPDATE operations applied to the DCR choreography events and relations [4, 7]. We only focus on these change operations as that they are challenging by themselves and that any change to a process can be written as a combination of these operations [13]. The on/off-chain separation ensures (i) the privacy of the partners as private information in private processes is not shared and kept off-chain and (ii) trust as the blockchain provides an immutable history of execution logs attesting the enforcement of correct execution of the choreography interactions [14]. Besides, 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. To summarize, we complement the work in [14] with the following contributions:

- We augment the SC managing the choreography process with change management techniques to enable change operations on a deployed instance of a choreography.
- We propose a protocol that allows (i) partners to first negotiate the change on-chain, (ii) then to dynamically update the choreography process instance managed by the SC with the new process change information, and (iii) finally propagate this information across partners processes affected by the change.
- We leverage the platform in [14] to integrate change to running DCR choreography instances.

The remainder of this paper is organized as follows. Sect. 2 presents fundamental definitions used in the approach and introduces an illustrating example. Sect. 3 reviews the main known related work. Sect. 4 details our approach. Sect. 5 presents an implemented prototype and some evaluation tests. Finally, Sect. 6 concludes the paper and gives insights into future work.

2 Basic Concepts and Illustrating Example

DCR graphs are one of many declarative business process modeling languages whose formalism is presented in [6]. A DCR graph G is represented by a triplet (E, M, Rel). E is a set of labelled *events*. M denotes the *marking* of the graph and is represented by the triplet (currently included events In, currently pending responses Pe, previously executed events Ex). Finally, Rel is the set of *relations* of the graph. Relations are of five types: condition$\longrightarrow \bullet$, response$\bullet \longrightarrow$, milestone$\longrightarrow \diamond$, include$\longrightarrow +$ and exclude$\longrightarrow \%$.

A **DCR choreography** models interactions between partners. Its execution is done in a distributed way. A DCR choreography is defined as follows [14]:

Definition 1. 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. A **public DCR process** of one partner represents the **projection** of the DCR choreography over this partner (see definition 4 in [14] for more details). The **private DCR process** of one partner 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.

Fig. 1 presents the trip e-booking scenario that was initially presented in the introduction (c.f. Section 1) translated into DCR: Fig. 1(a) presents the DCR choreography and Fig. 1(b) presents 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 a choreography interaction sent by Tourist and received by TouristOfficer. It is managed on-chain (c.f. [14]). To execute the send event, Tourist triggers the SC from its private DCR process. Table 1 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 and choreography 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. 1(b).

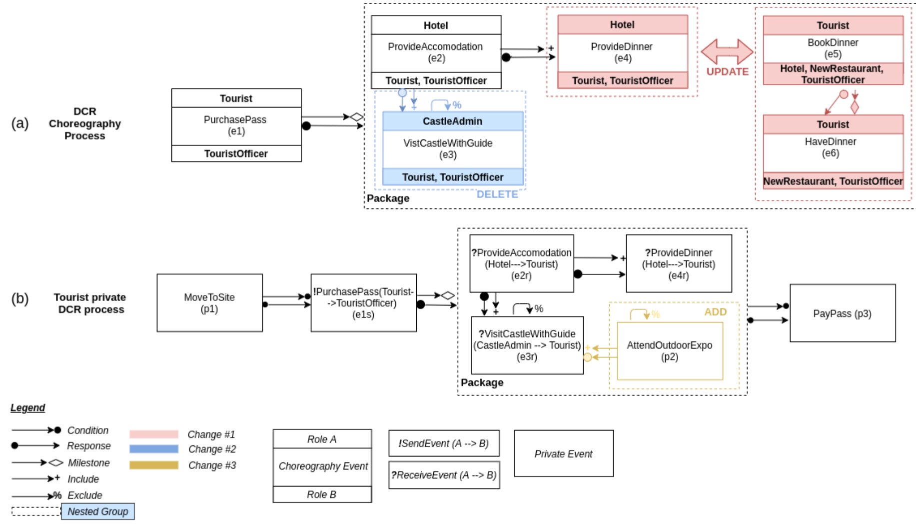

Fig. 1. DCR choreography process and tourist private DCR process of the trip e-booking process

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

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. 1(b), 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. 1, change #3 consists into adding the DCR fragment ($p2$: (1,0,0), $p2 \longrightarrow \bullet e3, p2 \longrightarrow +e3, p2 \longrightarrow \%p2$).

Definition 3. Change operation To define the change operations, we refer to [4] 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 2**. Change operations are of three major types[6]:

- $C \oplus G_{Ref}$ 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* 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. 1. 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 [5, 6].

In the following section, we present the related work regarding change mechanisms in cross-organizational business processes.

3 Related Work

Change management at runtime in procedural processes has been studied in [7] where change propagation algorithms ensure behavioral and structural sound-

[6]We use the same notation of the operations defined in [4]

ness of choreography partners private processes after the change. In [15, 8], authors consider the change negotiation phase but no mechanism is proposed to ensure that all partners have trustfully applied the change, and no blockchain is used to deal with this problem.

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 [4]. 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 [19], authors use a *correct-by-construction* approach on running instances of DCR graphs. The structure underlying a DCR is a labelled transition system. 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 [5], 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 [17], 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 [4, 5]. Additionally, approaches binding actors to roles in a process collaboration [17] 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 soundness is dealt with, the proposed approach does not provide a mechanism that ensures choreography partners project the change and propagate it trustfully.

4 Proposed Approach

DCR business processes monitored in the blockchain are represented into SCs as follows (c.f.[14]): the SC holds a set of activities, each assigned to an actor, and linked to an execution state. A relation matrix which summarises the execu-

Table 2. 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

tion 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 Sect. 4.1). 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 Sect.4.2 4.2). 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 Sect.4.3), followed by a propagation stage (step 4 in Sect.4.4.)

4.1 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 of the updated graph. A DCR graph is correct 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 [5]. We divide these rules in rules describing (i) allowed change rule (AR) and (ii) denied change rule (DR) presented in Table 2 [7].

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

[7] The reader can check [5] for more details about denied and allowed change operations in DCR orchestrations that inspired the proposed changes.

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*.

4.2 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[8]. 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.

Algorithm 1 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

[8] 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/).

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
 1 Function requestChange(h_curr, h_req, E, t1, t2):
 2   require msg.sender belongs to the list of business partners;
 3   if changeRequests[h_curr].status == Init then
 4     set changeRequests[h_curr].h_req ← h_req;
 5     set changeRequests[h_curr].status ← "BeingProcessed";
 6     set changeRequests[h_curr].initiator ← msg.sender;
 7     set changeRequests[h_curr].endorsers ← E;
 8     set changeRequests[h_curr].t1 ← t1;
 9     set changeRequests[h_curr].t2 ← t2;
10     emit RequestChange(h_curr, h_req, E, msg.sender);
11   else
12     emit Error; // an ongoing change request is being processed
13 End Function
```

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 [16]. 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. 1, 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 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]

4.3 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. 2. 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

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
         ∈ {0,1}
 1 Function endorserRSP(h_curr, e_s, rsp):
 2     require(block.timestamp <= changeRequests[h_curr].t1);
 3     require(e_s ∈ E);
 4     require(changeRequests[h_curr].changeEndorsement[e_s] != 1);
 5     require(changeRequests[h_curr].status == ("BeingProcessed");
 6     if rsp == 1 then
 7         set changeRequests[h_curr].changeEndorsement[e_s] ⟵ 1;
 8         emit AcceptChange(h_req, e_s);
 9         lockInstanceChecker(h_curr)
10     else if rsp == 0 then
           // declineapprovalOutcomes
11         set changeRequests[h_curr].status ← "Declined";
12         emit DeclineChange(h_req, e_s);
13     else
14         emit Error(h_req, e_s);
15 End Function
```

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 the SC checks whether the instance needs to be locked (line 9). The $lockInstanceChecker$ function assesses whether all endorsers have accepted the change: the $changeEndorsement$ 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.

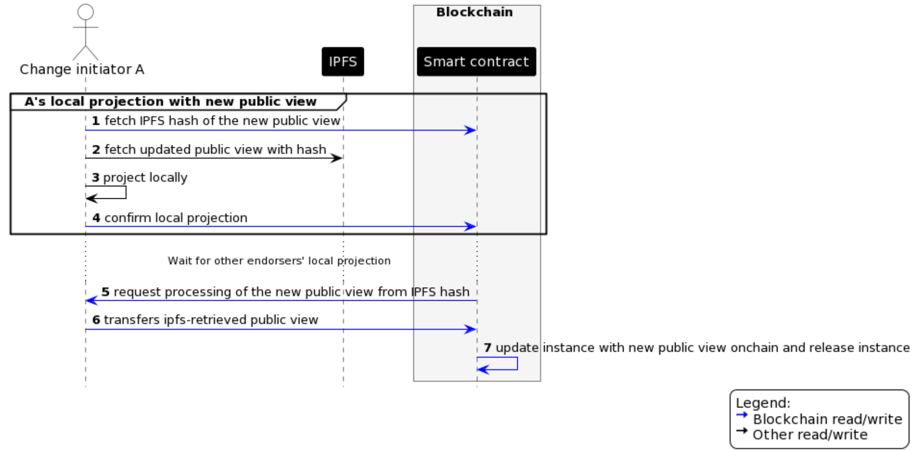

Fig. 2. Sequence diagram of the propagation stage illustrating the interactions between partners and the SC with A being the change initiator

4.4 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. To ensure the correctness of the change propagation, 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 $r' \in E$:

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 $\forall r \in E$.

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 Sect. 4.1). 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. The SC enforces propagation correctness 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. This is out of the scope of the present work and will be done in future work.

Indeed, Fig. 2 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. 2 step 1-4). Participant A first fetches the IPFS hash of

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
1  Function confirmProjection(h_curr, e_s):
2  |  require(e_s ∈ E);
3  |  require(block.timestamp <= changeRequests[h_curr].t2);
4  |  require(changeRequests[h_curr].didPropagate[id] != 1);
5  |  set changeRequests[h_curr].didPropagate[id] ⟵ 1;
6  |  emit LogWorkflowProjection(h_curr);
7  End Function
```

the new public view 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 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. 2 step 5) and forwards it to the SC (Fig. 2 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. 2 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 view 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 following the same approach as in [14]. 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 (c.f. [14] for a more detailed description).

5 Implementation and evaluation

5.1 Implementation

In [14], authors 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 [9].

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[10]. 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 (cf. [14]). 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 4.1: 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.

5.2 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 [10], 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.

[9] Code of the implemented prototype augmented with change management is accessible at https://github.com/tiphainehenry/adaptiveChangeDCR/

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

Table 3. 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

Table 3 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).

6 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. The SC records partners' involvement in a tamper-proof fashion during the change negotiation and propagation stages. If a misbehavior occurs, the blockchain logs can be used as a shared source of truth.

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 only consider the compatibility checks between public DCR processes of partners as a correctness criterion. This ensures that the DCR choreography is safe and terminates in acceptable state, i.e., is deadlock free after the change. We are currently working on proving the consistency checks between one partner's private and public DCR processes. In the present work, we focus on the current instance of the process. Nonetheless, it is also interesting to consider the change at the process model level and that after change is validated, all future instances follow the change.

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 [17]. 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 [21]. 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. van der Aalst, W.M., Weske, M.: The p2p approach to interorganizational workflows
2. Weber et al., I.: Untrusted business process monitoring and execution using blockchain. In: BPM. pp. 329–347. Springer (2016)
3. Madsen et al., M.F.: Collaboration among adversaries: distributed workflow execution on a blockchain. In: FAB. p. 8 (2018)
4. Mukkamala et al., R.R.: Towards trustworthy adaptive case management with dynamic condition response graphs. In: EDOC. pp. 127–136. IEEE (2013)
5. Debois et al., S.: Replication, refinement & reachability: complexity in dynamic condition-response graphs. Acta Informatica **55**(6), 489–520 (2018)
6. Hildebrandt et al., T.: Declarative choreographies and liveness. In: FORTE. pp. 129–147. Springer (2019)
7. Fdhila et al., W.: Dealing with change in process choreographies: Design and implementation of propagation algorithms. Information systems **49**, 1–24 (2015)
8. Fdhila et al., W.: Multi-criteria decision analysis for change negotiation in process collaborations. In: EDOC. pp. 175–183. IEEE (2017)
9. Van Der Aalst et al., W.M.: From public views to private views–correctness-by-design for services. In: WS-FM. pp. 139–153. Springer (2007)
10. Buterin, V., et al.: A next-generation smart contract and decentralized application platform. white paper **3**(37) (2014)
11. Debois, S., Hildebrandt, T., et al.: Safety, liveness and runtime refinement for modular process-aware is with dynamic sub processes. In: FM (2015)
12. Decker, G., Weske, M.: Behavioral consistency for b2b process integration. In: CAISE. pp. 81–95. Springer (2007)
13. Hallerbach, A., Bauer, T., Reichert, M.: Capturing variability in business process models: the provop approach. J SOFTW MAINT EVOL-R **22**(6-7), 519–546 (2010)
14. Henry, T., Brahem, A., Laga, N., Hatin, J., et al.: Trustworthy decentralized execution of declarative business process choreographies. In: ICSOC. Springer (2021)
15. Indiono, C., Rinderle-Ma, S.: Dynamic change propagation for process choreography instances. In: OTM Conferences. pp. 334–352. Springer (2017)
16. Li, C., Palanisamy, B.: Decentralized privacy-preserving timed execution in blockchain-based smart contract platforms. In: HiPC. pp. 265–274. IEEE (2018)
17. López-Pintado, O., Dumas, M., García-Bañuelos, L., et al.: Controlled flexibility in blockchain-based collaborative business processes. Information Systems (2020)
18. Mendling, J., Weber, I., Aalst, W.V.D., et al.: Blockchains for business process management-challenges and opportunities. ACM TMIS (2018)
19. Nahabedian, L., Braberman, V., D'ippolito, N., Kramer, J., Uchitel, S.: Dynamic reconfiguration of business processes. In: BPM. pp. 35–51. Springer (2019)
20. Ryu, S.H., Casati, F., Skogsrud, H., Benatallah, B., Saint-Paul, R.: Supporting the dynamic evolution of web service protocols in soa. TWEB (2008)
21. Xiong, F., Xiao, R., et al.: A key protection scheme based on secret sharing for blockchain-based construction supply chain system. IEEE Access (2019)
22. Xu, X., Weber, I., Staples, M.: Blockchain patterns. In: Architecture for Blockchain Applications, pp. 113–148. Springer (2019)