Change Propagation in Collaborative Processes Scenarios

Walid Fdhila, Stefanie Rinderle-Ma
University of Vienna, Austria,
Faculty of Computer Science
{walid.fdhila, stefanie.rinderle-ma}@univie.ac.at

Manfred Reichert
University of Ulm, Germany,
Institute of Databases and Information Systems
manfred.reichert@uni-ulm.de

***Abstract*—Process flexibility and change constitute major challenges for process-aware information systems. This does not only hold for centralized process scenarios, but also for collaborative ones involving multiple distributed and autonomous partners. If one partner adapts its private process, the applied change might affect the processes of the other partners as well. Hence the change must be propagated to concerned partners in a transitive way. A fundamental challenge is then to find ways of propagating the changes in a decentralized manner. Existing approaches dealing with changes of collaborative processes are limited with respect to the change operations considered and their dependency on certain process specification languages. By contrast, this paper presents a generic change propagation approach based on the Refined Process Structure Tree. Our approach is applicable independently of a particular process specification language. Further, it considers a comprehensive set of change patterns. Finally, it is shown that the provided change propagation algorithms preserve structural dependencies for any change pattern.**

I. Introduction

Process flexibility and change have been identified as major challenges in process management research for more than a decade [1], [2], [3]. Driving forces necessitating the adaptation and evolution of implemented processes include, for example, changes in law, exceptional situations, and process optimization. Change support is not only crucial for centralized processes (i.e., processes running entirely within one organization), but also for collaborative process scenarios involving multiple partners [4]. Typically, these partners run their own private processes and provide public views on them (i.e., local choreography models) to partners. Based on these public views, a global choreography can be formed by exchanging messages in a controlled manner in order achieve some common goal. A characteristic example of a collaborative processes is a supply chain consisting of the three partners *customer*, *producer*, and *supplier* [4]. Based on their internal or private processes, these partners provide local choreography models to the outside communicating through interactions.

Centralized processes pose many challenges when changes have to be applied to them. Existing approaches have focused on aspects such as correctness of change [2], change reuse [5], and performance [6]. The same challenges hold for collaborative process scenarios. In fact, changes of collaborative processes are by far more difficult to handle due to the non-availability of the information required to perform, for example, correctness checks. Specifically due to the absence of a central coordinator, such as assumed in previous work [7], information on the partners' private processes is not available to others aggravating the correct application of process changes. For example, assume a collaborative supply chain scenario as outlined above. Let further a change become necessary at the supplier's side due an emerging internal policy and assume that an additional notification message is required from the producer and - in the sequel - from the customer. This change would be first applied to the supplier's internal process. However, since the change affects both partners, it has to be *propagated* to the respective partner processes as well. In turn, this change propagation raises challenging issues: (a) Which information must be propagated to partners? (b) How to adapt the local choreographies in order to comply with the change? (c) How to adapt the partners' private processes? (d) How to maintain correctness and consistency of the global choreography as well as compatibility of the local choreography models among each other?

So far, only few approaches have addressed these challenges [4]. All these approaches are restricted with respect to the formal specification language used and the set of change operations considered. Furthermore, only the effects on direct partners have been investigated so far, factoring out issues such as transitive changes as well as issues related to the compatibility of the local choreography models as well as the consistency of the global choreography after change propagation. In this paper, we tackle change propagation in collaborative process scenarios in a more holistic way. First of all, we base our considerations on the Refined Process Structure Tree (RPST) [9]. RPST provides a process representation that is independent of any specific process specification language on the one side and enables us to formally specify and propagate changes on the other side. Regarding process changes, we investigated well-known process change patterns [10] and provide propagation algorithms for the most common change operations such as insert, delete or replace process fragments, and – at a more semantic level – update of, for example, input/output parameters. All considered change patterns operate on process

The work presented in this paper has been conducted within the project I743 funded by the Austrian Science Fund (FWF) and by the Deutsche Forschungsgemeinschaft (DFG).

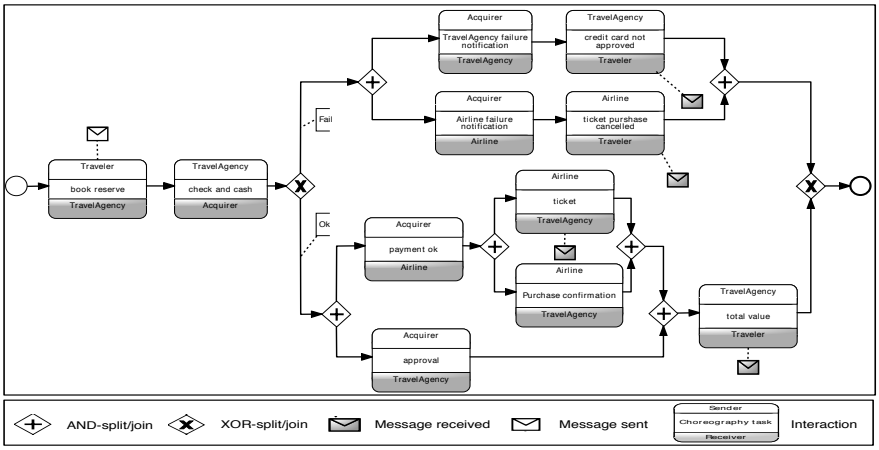


Fig. 1. Global Choreography: Book Trip Operation [8]

fragments. The associated propagation algorithms are formally defined and illustrated based on a use case. Furthermore, we show that the change propagation results in compatible local choreography models as well as a consistent global choreography. Finally, we sketch the transition from the structural considerations provided in this paper to semantic issues and future challenges.

In Sect. 2 the problem space is introduced, followed by fundamental definitions in Section 3. Section 4 provides the change propagation algorithms. Our approach is evaluated in Section 5. In Section 6 we discuss related work and Section 7 summarizes the paper.

II. Motivation and Challenges

We illustrate change propagation issues along the *booking* trip choreography example depicted in Figure 1. This example is part of the choreography model described in [8]. It is modeled in BPMN 2.0 and describes a collaboration between four partners, i.e., *traveler*, *travel agency*, *acquirer*, and *airline*. The *traveler* sends booking information to the *travel agency* which, in turn, contacts the *acquirer* to request a credit check. If the *traveler* has not enough credit, failure notifications are sent to the *travel agency* and the *airline* partners which inform the *traveler* about the reservation failure and the purchase cancellation respectively. Otherwise, an approval is sent to the *travel agency* and the *airline* is triggered to send the ticket and the purchase confirmation.

Figure 2 depicts the public views (i.e., local choreography models) of all participants involved in the choreography. Each partner view includes only the interactions in which this partner is involved as well as the control flow between them. Figure 2 does not show the private view (i.e., orchestration) of each partner that includes the internal activities. These local views merged together lead to the global choreography model. Now assume that partner *acquirer* wants to change the logic of its private process. For instance, instead of sending two failure notifications to *airline* and *travel agency* respectively, he notifies only one of them. In Figure 2, fragment $\mathcal{F}$ is replaced by fragment $\mathcal{F}\prime$ (XOR instead of AND). Then, for some process instances, partner *airline* would wait indefinitely for a failure notification if the *acquirer* chooses to send a notification only to the *travel agency*. In this case, the *acquirer* process instance would be blocked in this state and not terminate. Hence, the described change affects the soundness of the choreography model in terms of compatibility between the distributed public views. To overcome this problem, the effects of this local change in the *acquirer* process model should be propagated to the concerned partners and the interactions should be restructured consequently.

Generally, the challenges are as follows: How to compute the impacts of this local change on the other partners and the global choreography? How to propagate the effects of the local change while preserving the soundness of the collaboration in terms of consistency and compatibility [11]? How to deal with transitive effects of changes? What if changes are made dynamically, i.e. during the execution of choreography instances? How to manage the different versions of the choreography model (used in monitoring) and the participants' processes at runtime?

III. Preliminaries

As emphasized, this paper deals with change propagation in *collaborative processes*. In practice, there are three different, but overlapping viewpoints in a collaboration: orchestration, behavioral interface, and choreography [12]. An *orchestration* describes the internal business logic of private processes as well as the communication actions in which a particular partner engages. The *behavioral interface* (also called *public view* or *local choreography model*) sketches the message exchanges from the perspective of a single partner as well as the sequencing between them. In other words, it represents an abstraction of an orchestration's internal actions. The *choreography* (also called *global model*) gives a global view of all interactions in the collaboration. It captures the interactions in which the participating partners engage to achieve a goal as well as the dependencies between these interactions. In order to precisely define the notions of orchestration, behavioral interface and choreography, we

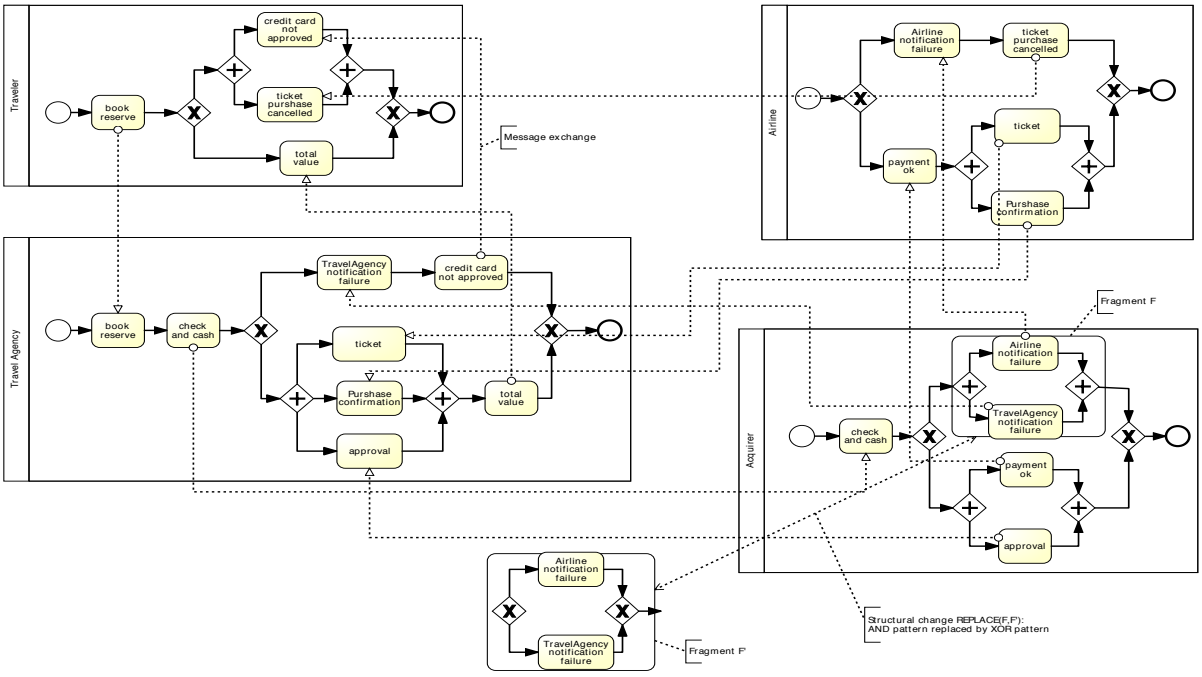


Fig. 2. Book Trip Operation: Local Views

need to adopt a model for representing control-flow relations between activities and interactions. In this paper, we adopt a structured representation of process models. In essence, a process model is represented as a tree whose leaves represent activities or interactions and whose internal nodes represent either sequence (SEQ), parallel (PAR), choice (CHC), or repeat loop (RPT) constructs. Structured process models are very close to BPEL. As an advantage they are simpler to analyze and easier to comprehend. Although it is possible to express unstructured models both when using BPEL and BPMN, recent work has shown that most unstructured process models can be automatically translated into structured ones [13].

Definition 1: [*(Structured) Process Model*] A process model π_p of partner p (private view) is a tree with the following structure (we use the type definition syntax of the ML language [14]):

$$\begin{aligned}
Process &::= Node \\
Node &::= Activity | ControlNode | Event \\
Activity &::= InternalActivity | InteractionActivity \\
InteractionActivity &::= Send(Message, Destination) \\
&\quad | Receive(Message, Sender) \\
ControlNode &::= SEQ([Node]) | CHC(\{Node\}) \\
&\quad | PAR(\{Node\}) | RPT(Node) \\
Event &::= Start \ | End
\end{aligned}$$

In this structuring of a process model, we consider an interaction activity *one-to-many-send* (respectively *one-from-many-receive*) as several interaction activities of type $send$ (respectively $receive$) executed in parallel. We define a ***fragment*** as a non-empty subgraph of a process model with a single entry and a single exit edge. In Definitions 2 and 3, we refer to the same *control nodes* and *events* as in Definition 1.

Definition 2: [*Local Choreography Model*] A local choreography model $\mathcal{L}_p$ of partner p states the external behavior of p and is also denoted public view. It includes the interactions with other partners as well as the constraints between them from the view point of this partner:

$$\begin{aligned}
LocalModel &::= Node \\
Node &::= Send(Message, Destination) \\
&\quad | Receive(Message, Sender) \\
&\quad | ControlNode | Event
\end{aligned}$$

Definition 3: [*Global Choreography Model*] The global choreography model $\mathcal{G}$ represents a global view of the interactions between collaborating partners.

$$\begin{aligned}
GlobalModel &::= Node \\
Node &::= I(Source, Destination, Message) \\
&\quad | ControlNode | Event
\end{aligned}$$

where I corresponds to interaction between the partners $source$ and $destination$. We consider $\mathcal{P}$ as the set of all partners participating in the choreography, Π as the set of their process models, and $\mathcal{L}$ as the set of their local models.
Figure 3 depicts the RPST model of the book trip operation's global choreography model as presented in Figure 1. Leaves represent interactions and internal nodes represent the control flow patterns used.
In order to support changes of collaborative processes, we consider basic change operations since more complex change patterns can be expressed by combining of these operations.

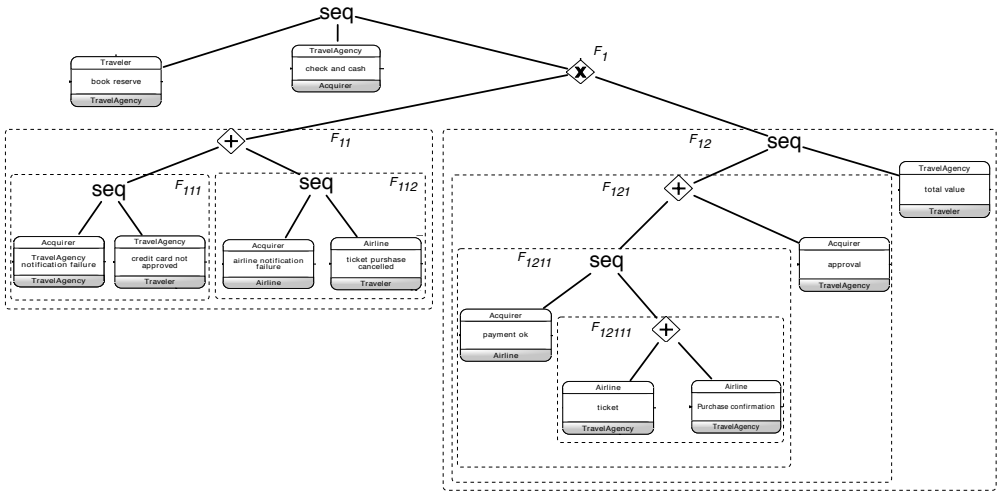


Fig. 3. Book Trip Operation: Refined Process Structure Tree

Change operations are defined as follows.

Definition 4: [*Change Patterns*]

$$
\begin{aligned}
ChangePattern &::= Structural \mid Semantical \\
Structural &::= REPLACE(oldFragment, newFragment) \\
&\quad \mid DELETE(fragment) \\
&\quad \mid INSERT(fragment, how, in, out) \\
how &::= Parallel \mid Choice \mid Sequence \\
Semantical &::= UPDATE(activity, attribute, newValue) \\
attribute &::= partner \mid role \mid Input \mid Output
\end{aligned}
$$

where REPLACE replaces an existing fragment with a new one, DELETE removes an existing fragment, INSERT adds a new fragment into the process model between in and out, and UPDATE modifies an attribute of a single activity.

Definition 5: [*Change Operation*] A change operation is a tuple (δ,Σ) where δ is the change pattern and Σ is either the process model, or the local or global choreography model to be changed.

Definition 6: [*Abstraction function*] Abstraction function $abstr_{\lambda}(\pi)$ is a projection of a process model π according to criterion λ. It transforms the source process model π into a target model π' where π' includes only activities satisfying λ (e.g., the local choreography model is an abstraction of its respective process model according to $interaction\ activities$). A reduction function may be applied to eliminate unnecessary control patterns after abstraction (e.g., a parallel pattern between an activity and an empty transition may be reduced to sequence) [15], [16]. For instance, depending on a certain condition, a different value of a data element d is sent to the same partner p. In the local view, this is reduced to only one interaction which corresponds to the sending of data d.

Next, we extend Definition 6 to consider fragments as well as local and global choreography models. We also assume that $\lambda = p\prime$ means the interactions with $p\prime$. Hence, $abstr_{p'}(\mathcal{L}_p)$ is the abstraction of $\mathcal{L}_p$ according to the interactions with p'.

Definition 7: [*Complement function*] If $a \in p$ is an interaction activity with a partner $p\prime$, the complement of a called $\overline{a} \in p\prime$ is the opposite of a as follows.

- $\overline{send(message, p\prime)} = receive(message, p)$.
- $receive(message, p\prime) = send(message, p)$.

If $\mathcal{F}$ is a fragment that solely includes interaction activities, $\overline{\mathcal{F}}$ corresponds to a fragment with same structure where each activity is replaced by its complement.

Lemma 1: $abstr_{p\prime}(\mathcal{F}) \in \mathcal{L}_p \implies \overline{abstr_{p\prime}(\mathcal{F})} \in abstr_p(\mathcal{L}_{p\prime})$
The complement of the abstraction of a participant fragment $\mathcal{F} \in \mathcal{L}_p$ according to participant $p\prime$ is a fragment of the abstraction of $\mathcal{L}_{p\prime}$ according to p.

Proof: The proof of this lemma can be based on the following compatibility properties of the collaboration [11].

- If $a \in \mathcal{L}_p$ is an activity that interacts with partner $p\prime$, then: $\exists b \in \mathcal{L}_{p\prime}$ with $b = \overline{a}$.
- If a_i, $a_j \in \mathcal{L}_p$ are two activities that interact with the same partner $p\prime$ and $\beta(a_i, a_j)$ is a function that returns the minimal precedence relation between a_i and a_j (after reduction), then: $\exists b_i, b_j \in \mathcal{L}_{p\prime}$ with $b_i = \overline{a_i}$, $b_j = \overline{a_j}$ and $\beta(a_i, a_j) = \beta(b_i, b_j)$ (bi-simulation property [11], [13]) .

■

Definition 8: [α *function*] Let π be a process model and $\mathcal{F}$ be a fragment with: $\forall o \in \mathcal{F} \implies o \in \pi$, where o denotes an object of type activity or control node. Then: Function $\alpha_{\pi}(\mathcal{F})$ returns the smallest fragment in the process structure tree of π that contains all elements of $\mathcal{F}$. This function is also used to identify a fragment in a local or global choreography model.

IV. Change Propagation

This section introduces a global view of the proposed approach and details the different steps to propagate changes in collaborative processes.

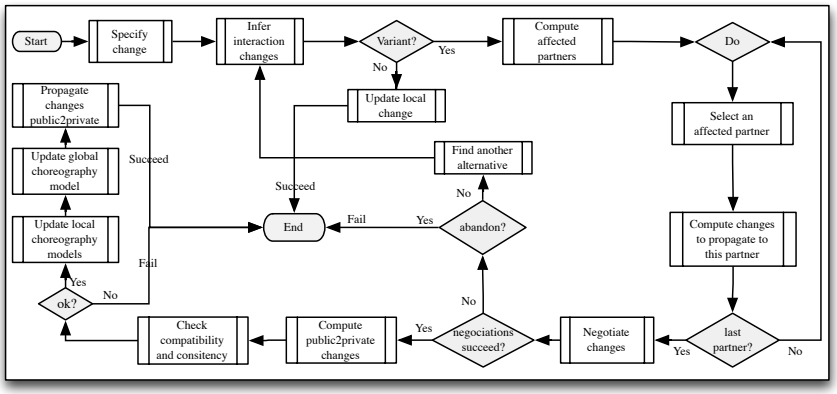


Fig. 4. Change Propagation - Major Steps

A. Overview

As afore mentioned, our objective is to support change propagation in choreographies with multiple interacting partner processes. Our approach relies on four major steps: (i) identifying interactions affected by the change as well as the partners involved, (ii) translating the initial change operation into several change operations each related to a particular partner, (iii) negotiating with the affected partners, and (iv) checking consistency and compatibility of the changed choreography. Figure 4 details the previous steps and outlines the different actions to achieve a sound propagation.

Given a change operation on one partner orchestration, we extract all interaction activities concerned by the change. Interaction activities are communication actions with other partners (i.e., sending and receiving requests). If the list is empty (i.e., if the local change is restricted to the internal behavior), the other business partners are not affected by the change and there is no need for a new agreement on the global choreography. Otherwise, the list of affected interactions is analyzed to identify all partners involved. For each of these partners, a relative change computation is undertaken to gather the changes to be propagated. The latter are computed according to the change operation type. Then, a negotiation phase is launched with each affected partner. If all negotiations succeed, we apply consistency and compatibility checks to ensure the soundness of the obtained models. If these models are sound, we update the local and the global choreography models affected by the change and, if necessary, adapt concerned orchestrations to their new behavioral interfaces. Section IV-B is structured according to the mentioned steps. It sketches the different algorithms needed for propagating changes. Note that this propagation strongly depends on the change operation type (i.e., INSERT, DELETE, REPLACE and UPDATE).

B. Algorithms Enabling Change Propagation

Consider a change operation δ on a process model π_p. To start, we abstract the changed fragment according to its interaction activities; i.e., we look for the structural or semantical changes that affect the interactions with the other partners. According to the change operation type, a particular procedure for decomposition is performed. This decomposition transforms the initial change operation into several change actions to be propagated to each affected partner. Due to lack of space, here, we only consider the UPDATE and the REPLACE change patterns. Note that REPLACE can cover patterns INSERT and DELETE.

1) REPLACE Decomposition: Assume we want to replace a fragment $\mathcal{F}$ by a new fragment $\mathcal{F}'$ with a different structure (e.g., in the *book trip* example depicted in Figure 1; $REPLACE(PAR(Airline\_notification\_failure, TravelAgency\_notification\_failure), CHC(Airline\_notification\_failure, TravelAgency\_notification\_failure)), \pi_{acquirer})$). As described in Algorithm 1, the first step computes all partners involved in $\mathcal{F}$ and $\mathcal{F}'$ (e.g., $Airline$ and $TravelAgency$) and for each partner p' we abstract from, the respective interaction activities ($abstr_{p'}(\mathcal{F})$ and $abstr_{p'}(\mathcal{F}')$) (e.g., in the example, $\mathcal{F}$ and $\mathcal{F}'$ already include interactions activities only). At this level, for a particular partner, three possible scenarios can be identified: **(i)** a partner involved in the original fragment $\mathcal{F}$ is no longer present in the new fragment $\mathcal{F}'$ (deletion of the interaction with this partner), **(ii)** a partner involved in $\mathcal{F}'$ was not present in $\mathcal{F}$ (adding a new interaction), and finally **(iii)** a partner present in both fragments $\mathcal{F}$ and $\mathcal{F}'$, but with different structure (updating the conversation with a partner). As a consequence of these scenarios, the REPLACE pattern is translated into a concatenation Δ of change patterns to propagate to the affected partners.

(i) Deletion scenario: If a particular partner p' interacts with the partner p in fragment $\mathcal{F}$ and is not engaged in any conversation with p in the new fragment $\mathcal{F}'$, we delete the respective interactions from their local choreography models. To achieve this, we consider each interaction activity a with p' to be deleted in $\mathcal{F}$. Further, we search for the matching activity $\overline{a}$ in p' and derive the change pattern DELETE($\overline{a}, \mathcal{L}_{p'}$). Note that this operation is not applied directly since there is a

negotiation phase. Besides, this deletion could have subsequent effects on p'. Indeed, it is possible that other interactions in p' semantically depend on the deleted interaction (e.g., supply chains scenarios). This is called *transitivity effect* and it does not constitute a structural problem, but a semantical one. In this section, we focus only on structural changes, but resolve the transitivity effects separately in Section IV-C. So, if activities to delete are synchronous (e.g., waiting for a response or replying to a request), then we delete the corresponding activities. For example, consider a *buyer* that invokes a *seller* for a price request (i.e., *send(seller, price_request), receive(buyer, price_request)*). If we delete this interaction, we must delete the corresponding response to the *buyer* as well (i.e., *send(buyer, price_response), receive(seller, price_response)*). This could be achieved using the correlations between activities (e.g., correlation sets known from BPEL).

Algorithm 1: Decomposition of Change Pattern $REPLACE_\pi(\mathcal{F}, \mathcal{F}')$

Input: - Set of all local choreography models $\mathcal{L} = \{\mathcal{L}_p | p \in \mathcal{P}\}$
- Global choreography model $\mathcal{G}$
begin
  $\Delta \leftarrow \emptyset$ //decomposition result
  $\mathcal{P}_\Delta \leftarrow$ List of all business partners involved in $\mathcal{F} \cup \mathcal{F}'$
  for *(each partner* $p \in \mathcal{P}_\Delta$*)* **do**
    $\mathcal{F}_\mathcal{G} \leftarrow \alpha_\mathcal{G}(\mathcal{F})$ {smallest fragment including $\mathcal{F}$ in $\mathcal{G}$}
    if $abstr_p(\mathcal{F}) \neq abstr_p(\mathcal{F}')$ **then**
      if $abstr_p(\mathcal{F}) = \emptyset$ **then**
        $in = T\_preset_p(\mathcal{F}_\mathcal{G}, \mathcal{G})$
        $out = T\_postset_p(\mathcal{F}_\mathcal{G}, \mathcal{G})$
        if *(*$\exists a \in \mathcal{L}_p$ *between* in *and* out*)* **then**
          $\Delta \leftarrow \Delta \wedge INSERT_p(\overline{abstr_p(\mathcal{F}')}, Parallel, \mathcal{L}_p.in, \mathcal{L}_p.out)$
        else
          $\Delta \leftarrow \Delta \wedge INSERT_p(\overline{abstr_p(\mathcal{F}')}, Sequence, \mathcal{L}_p.in, \mathcal{L}_p.out)$
        end
      else
        if $abstr_p(\mathcal{F}') = \emptyset$ **then**
          for *(each* $a \in \overline{abstr_p(\mathcal{F})}$*)* **do**
            $\Delta \leftarrow \Delta \wedge DELETE_p(a)$ {reduction rules may apply when executing the changes}
            if *(*a *is synchronous)* **then**
              $\Delta \leftarrow \Delta \wedge DELETE_\pi(a') \wedge DELETE_p(\overline{a'})$ {a' and $\overline{a'}$ are the relative responses or requests of a and $\overline{a}$ }
            end
          end
        else
          $\Delta \leftarrow \Delta \wedge REPLACE_p(\alpha_p(\overline{abstr_p(\mathcal{F})}), \gamma(abstr_p(\mathcal{F}'), \alpha_p(\overline{abstr_p(\mathcal{F})})))$ {γ is the merge function}
          for *each* $a \in \mathcal{F}$ *st* $a \notin \mathcal{F}'$ **do**
            $\Delta \leftarrow \Delta \wedge DELETE_p(\overline{a})$ {if synchronous we do the same procedure as before}
          end
        end
      end
    end
  end
  Output Δ ;
end

(ii) Insertion scenario: If a particular partner p' has no interactions with p in the old fragment $\mathcal{F}$, but interacts with p in the new fragment $\mathcal{F}'$, we have to insert the new interactions in the local choreography model of p' ($\mathcal{L}_{p'}$). For this purpose, we abstract $\mathcal{F}'$ according to partner p', compute $\overline{abstr_{p'}(\mathcal{F}')}$ and lookup for the exact insertion position of the latter in $\mathcal{L}_{p'}$. In order to compute this position, while preserving structural dependencies, we make use of the global choreography model. Indeed, partner p' may have other interactions with p outside the changed fragment $\mathcal{F}$ or with other partners outside or inside $\alpha_\mathcal{G}(\mathcal{F})$ in the global model $\mathcal{G}$ (where $\alpha_\mathcal{G}(\mathcal{F})$ is the smallest fragment that encapsulates $\mathcal{F}$ in the RPST of $\mathcal{G}$). Note that $\mathcal{F}_\mathcal{G}$ includes all interactions of p which appear in $\mathcal{F}$ merged with other interactions of different partners. For example, consider the change scenario introduced in Section II, where fragment $\mathcal{F}$ is replaced by $\mathcal{F}'$. The smallest fragment containing the two activities of $\mathcal{F}$ (i.e., *airline* and *travel agency* failure notifications) in the RPST of the $book\ trip$ global choreography model is $\mathcal{F}_{11}$ (cf. Figure 3). The latter includes other interactions concerning the *airline* and *travel agency* partners (i.e., credit_card_not_approved and ticket_purchase_cancelled).

Algorithm 2: Transitive Preset and Postset Computation: $T\_preset_p(\mathcal{F}, \pi)$ $T\_postset_p(\mathcal{F}, \pi)$

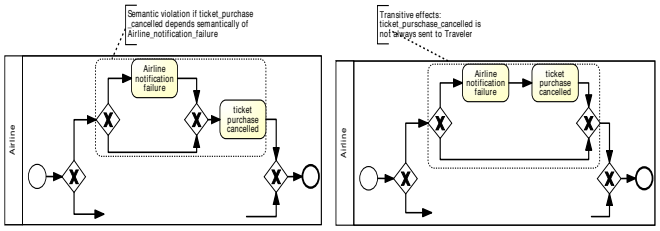
Init: - $T\_preset = start$
- $T\_postset = end$
begin
  $predecessor = \mathcal{F}.FirstElement$
  repeat
    $predecessor \leftarrow predecessor.parent$
    if $predecessor = sequence$ **then**
      repeat
        $childleft \leftarrow predecessor.Child.left.next$ {parse from right to left}
        $childright \leftarrow predecessor.Child.right.next$ {parse from left to right}
        if $\exists \mathcal{F}' \in childleft$ *st* $\exists a_i \in \mathcal{F}'$ *which invokes* p **then**
          $T\_preset = abstr_p(child)$
          $preset\_path$ = path between $\mathcal{F}$ and T_preset
        end
        if $\exists \mathcal{F}' \in childright$ *st* $\exists a_i \in \mathcal{F}'$ *which invokes* p **then**
          $T\_postset = abstr_p(child)$
          $postset\_path$ = path between $\mathcal{F}$ and $T\_postset$
        end
      until ;
    end
  until *(*$(T\_preset \neq start \wedge T\_postset \neq end) \vee predecessor =$ *rootElement)* ;
end
Output: $T\_postset$, T_preset ;

Once $\alpha_\mathcal{G}(\mathcal{F})$ is identified, we compute the transitive $preset$ and $postset$ of $\alpha_\mathcal{G}(\mathcal{F})$ according to p'. The transitive postset $T\_postset_{p'}(\alpha_\mathcal{G}(\mathcal{F}), \mathcal{G})$ (preset $T\_preset_{p'}$ $(\alpha_\mathcal{G}(\mathcal{F}), \mathcal{G})$) represents the smallest fragment of $\mathcal{G}$ containing interactions of p' that could be executed just before (just after) $\alpha_\mathcal{G}(\mathcal{F})$. These two variables are used to identify the insertion position of $\overline{abstr_{p'}(\mathcal{F}')}$ in $\mathcal{L}_{p'}$. To compute these values we refer to Algorithm 2. Given a fragment $\mathcal{F}$, a process model π, and a partner p, $T\_preset_p(\mathcal{F})$ is calculated as follows: Algorithm 2 first parses the RPST from $\mathcal{F}$ to the root element, and for each $sequence$ element found, it parses the child elements from the right to the left and checks whether a child contains activities or interactions with p. If this applies, $\overline{abstr(child)}$ represents the transitive preset. If no child element contains

interaction activities with p, we try to discover the upper level in the tree and repeat the same procedure. The postset is computed the same way except that we discover the right part of the process tree. The transitive preset and postset ($in = T\_preset_{p'}(\ \alpha_{\mathcal{G}}(\mathcal{F}), \mathcal{G})$ and $out = T\_postset_{p'}(\ \alpha_{\mathcal{G}}(\mathcal{F}), \mathcal{G})$) represent the insertion position of the fragment $\overline{abstr_{p'}(\mathcal{F}')}$ in $\mathcal{L}_{p'}$. If there are no activities between in and out in the initial model, we insert $\overline{abstr_{p'}(\mathcal{F}')}$ in sequence. Otherwise, we insert it in parallel.

(iii) Replacement scenario: Finally, the last scenario is when both fragments $\mathcal{F}$ and $\mathcal{F}'$ contain interactions with a partner p', but with different structures (control flow). This means that the dependencies between interactions have changed and therefore the local view of p' should be updated to conserve compatibility between the behavioral interfaces. For instance, in the change operation example from Figure 2, $\mathcal{F}'$ contains the same interaction with airline for sending the failure notification, but with different structure (the latter could be sent or not depending on the running instance). So, the idea is to abstract the interactions with p' in $\mathcal{F}$ ($\overline{abstr_{p'}(\mathcal{F})}$), identify $\overline{abstr_{p'}(\mathcal{F})}$ in $\mathcal{L}_{p'}$, and replace it with $\overline{abstr_{p'}(\mathcal{F}')}$. However, this is not possible if the smallest fragment in $\mathcal{L}_{p'}$, which contains the activities of $\overline{abstr_{p'}(\mathcal{F}')}$, includes other interactions with different partners. For this purpose, we adopt a merge algorithm to merge $abstr_p(\mathcal{F})$ with $\alpha_p(\overline{abstr_p(\mathcal{F}')})$. Indeed, merging process models has been widely studied and many works were proposed [17], [18]. The key idea is to merge different (and overlapping) process models into a single model without restricting the behavior that was possible in the original models. So, if we consider γ as a merge function then the problem could be solved by deleting the activities of $\overline{abstr_p(\mathcal{F})}$ from $\alpha_p(\overline{abstr_p(\mathcal{F})})$ and then merging $abstr_p(\mathcal{F}')$ with the rest of activities in $\alpha_p(\overline{abstr_p(\mathcal{F})})$. The merge could also lead to a semantic violation. For instance, if we refer to our change scenario for the book trip operation, the change propagation to the $airline$ partner is depicted in Figure 5. Two scenarios are possible: the modified fragment is inserted before activity $ticket\_purchase\_order$ or it is merged with it. The first case is structurally correct in terms of dependencies, and it conserves the compatibility between the interfaces. However, in case $ticket\_purchase\_order$ semantically depends of some parameters of activity $airline\_notification\_failure$, the second scenario is considered to be more correct. This is a semantic issue and should be validated by the target partner affected. The latter should validate the proposed scenario which goes with its semantics.

2) UPDATE Decomposition: As afore mentioned, the UPDATE pattern is used to update the attributes of a single activity. This paper considers the update of attribute $partner$, but this can be easily be generalized to consider other attributes as well (e.g., $input$, $output$, etc.). For instance, we assume that an interaction activity a of π_p should be updated to interact with p'' instead of p'. In this case, we look at $\overline{a}$ in $\mathcal{L}_{p'}$ and delete it. If the interaction is synchronous, we delete the



(a) first scenario (b) second scenario

Fig. 5. Airline: Change Propagation

corresponding activity to $\overline{a}$ (i.e., reply to a request or receive a response) correlating inputs and outputs. We also update the corresponding activity to a in $\mathcal{L}_p$ with the new partner p''. Then, we check whether or not p'' already exists in the list of partners. In the latter case, we create a new public view for p'' containing $\overline{a}$ and the corresponding activity $\overline{a''}$ if the communication is synchronous. If the partner p'' already exists, we check where to insert $\overline{a}$ and possibly $\overline{a'}$ in $\mathcal{L}_{p''}$. For this purpose, we use the global choreography model $\mathcal{G}$ and compute the transitive preset and postset of $\overline{a}$ (in and out) and $\overline{a'}$ (in' and out') according to p''. If there are other interactions with p'' between in and out, we compute their dependencies with $\overline{a}$ and insert the latter in $\mathcal{L}_{p\prime}$ accordingly. The same holds for $\overline{a'}$ if the communication is synchronous.

Algorithm 3: Decomposition of Change Pattern $UPDATE_\pi(a, attribute, newValue)$

```
1  Input: -Set of all business partners P
2         - Set of all local choreography models L = {L_i, i ∈ P}
3         - Global choreography model G
4  begin
5      if (attribute = partner) then
6          p' ← a.partner {oldpartner}
7          p'' ← value {newpartner}
8          if p'' ∉ P then
9              Create(l_p'') {new local view}
10             Add(l_p'', L) and Add(p'', P)
11         end
12         Δ ← Δ ∧ DELETE_p'(ā)
13         Δ ← Δ ∧ INSERT_p''(ā, Parallel|Sequence, in, out)  {in
           and out are computed as in previous algorithms}
14         if Synchronous then
15             a' = Response(ā)
16             Δ ← Δ ∧ DELETE_p'(a)
17             Δ ← Δ∧INSERT_p''(a', Parallel|Sequence, in', out')
               {in' and out' are computed as in previous algorithms}
18             if p' = ∅ then
19                 remove(p', P) and remove(l_p', L)
20             end
21         end
22     else
23     end
24 end
25 Output Δ ;
```

C. Delete pattern: dealing with transitivity

We now present a non-exhaustive list of use cases showing the transitivity effects of the DELETE change pattern and the solutions to cope with them. Note that this is a semantic issue and can not be resolved using the propagation algorithms presented so far and focusing on structural propagation.

For example, consider three partners p_1, p_2 and p_3 with p_1 invoking p_2, which in turn, invokes p_3. The latter returns the intermediary result to p_2, which applies data transformations and sends the final result to p_1. So, if p_1 decides to delete its interaction with p_2, we also must delete the subsequent interaction between p_2 and p_3 which is just used to deliver the final result. If a partner deletes an interaction, Semantically this means that he is not able to afford this service anymore, or he does not need the data anymore. The challenge is to determine if an interaction is transitive according to another one, to identify the transitivity effects, and to resolve them.

Case 1: If partner p is the ***final consumer*** of a data element; he launches a communication in which he requires a response. p includes correlated $send$ and $receive$ patterns S/R (non-blocking S/R pattern).

- p deletes the $send$/$receive$ patterns. This means that p does not need the data anymore (since p is the final consumer). We delete $\overline{send}$ and $\overline{receive}$. In case where all subsequent interactions with other partners are **just** used to deliver this data, we delete them (e.g. supply chain scenarios). It is possible that only a subset of the subsequent interactions are used to deliver this data. These are deleted only if they don't have any other role in the choreography.
  Example: two concurrent request from A and B to the same partner C. The latter does subsequent interactions and then reply to both of them. If A deletes his interaction, we should not delete the subsequent interactions of C since they are still used to reply to B.
- p deletes only the $send$ pattern. Two scenarios: **(i)** Another partner starts the communication instead of him and then we just Update the correspondent $\overline{send}$ with the new partner, or **(ii)** p is not responsible anymore for informing the other partner to start the processing of the response (the response starts automatically or under other constraints) and then we Delete the corresponding $\overline{send}$.
- p deletes only the $receive$ pattern. This means either he does not need the data anymore or the data is transferred to another partner. In the first case, we just delete the corresponding $\overline{receive}$ and look for other interactions correlated with this response (**just** used for delivering the response). In the second case, we update the corresponding $\overline{receive}$ with the new partner.

Case 2: Partner p is the **final consumer** of the data but is not responsible for launching the first interaction (p has only the $receive$). Then, if p deletes the $receive$ (does not need the data anymore), we delete the corresponding $\overline{receive}$ as well as all subsequent interactions used **only** to deliver this data. All interactions that participate in delivering this data but have another role in the choreography, are kept.

Case 3: p is the **starting point**, responsible for starting a communication resulting in a response to another partner (p has the $send$). Either **(i)** another partner is responsible for starting this communication; then, we update the $\overline{send}$ with the new partner, or **(ii)** the communication starts automatically or under other constraints on the target partner; then we delete the corresponding $\overline{send}$. Subsequent interactions are not deleted since we still need the final data to be delivered to the final consumer.

Case 4: p is an **intermediary partner** in the conversation.

- p has correlated $receive$ and $send$ in **sequence**. p receives a request, starts a **subsequent** interactions necessary for delivering the final response to the requester.
  - If p deletes the $send$ and the $receive$ patterns, this means that p is not able to afford the service anymore, but we still need the final response to the requester. In this case, we have two choices: **(i)** looking for another partner who can do the work of p to deliver this response; in this case, we update the subsequent interactions as well as the interactions of the root partner (the one who invoked p and the one yo which p should send the result) with this new partner, **(ii)** deleting $\overline{send}$ and $\overline{receive}$ as well as all subsequent interactions used **only** in the context of this **intermediary data** and looking for another partner or set of partner who fulfill this intermediary task. Then we update the communications with the root partners or p.
  - If p deletes only the $send$ pattern, this means another partner is responsible for starting this intermediary communication or the subsequent interactions start automatically or under other constraints. In the first case, we update the corresponding $\overline{receive}$ with the new partner, otherwise we just delete the $\overline{receive}$.
  - If p deletes only the $receive$ pattern, this means that p can not or do not want to do anymore the operations **just after** the $receive$ and related to the delivery of the final data. **(i)** If no related operations are necessary to deliver the intermediate data, we update the $\overline{receive}$ to link it directly with root partner **(ii)** If related operations are necessary, we look for another partner which can fulfill the same tasks.

V. Evaluation

In this paper, we assume that the initial public views of the collaborative processes are compatible with each others and that each private view is consistent with the respective behavioral interface definition (i.e., the local choreography model). We further assume the well-behavedness of the change operation in terms of structure and semantics. In [11], the authors distinguish between structural and behavioral compatibility.

- *Structural compatibility* requires that for every message that may be sent, the corresponding interaction partner is able to receive it. In turn, for every message that can be received, the corresponding partner must be able to send such a message
- *Behavioral compatibility* considers behavioral dependencies (i.e., control flow) between different message exchanges within one conversation.

In our propagation mechanism, structural compatibility is preserved. Indeed, according to the change operation type, for each affected partner we add, update, or remove the complement of what has been changed in the modified process (cf. **Lemma 1**). This means that for each $send$ pattern in one source partner there is the corresponding $receive$ pattern with the expected attributes in the process of the target partner.

Now, consider (δ, π_p) being the change operation to be applied to process model π_p and $(\delta, \mathcal{L}_p)$ the inferred change to be applied to the local view of p. Δ is the set of inferred changes from $(\delta, \mathcal{L}_p)$ to be propagated to its directly affected partners. For each affected partner p_i, $(\delta_i, \mathcal{L}_i)$ represents the inferred change operation to be propagated to its public view ($\Delta = \wedge_{i=1..n}(\delta_i, \mathcal{L}_i)$, where n is the number of affected partners). Note that the number of inferred structural changes is finite since we only consider propagations to direct partners. In turn, changes that might have structural effects on other partners (transitive relations) are propagated to them by their direct partners.
If $(\delta, \mathcal{L}_p)$ is invariant, i.e., it does not affect the behavioral interface of p, consistency and compatibility are preserved over the collaborative processes. In addition, since processes as well as the changed fragments are structured, consistency and compatibility relations are reduced to those between the fragments affected by the change.

- The INSERT pattern augments the process models of the partners affected by the change with new activities. Further, it does not affect the structural or behavioral relations between the existing activities. However, some direct precedence relations may be transformed into transitive ones. The decomposition of the INSERT pattern results only in INSERT patterns in the public views of the affected partners. According to one partner, the insertion is done with respect to the direct and transitive dependencies with the activities of the same partner. As explained in Section IV, if $\mathcal{F}$ represents the fragment to be inserted in $\mathcal{L}_p$, $\overline{abstr_i(\mathcal{F})}$ is the fragment to be inserted in $\mathcal{L}_i$. The latter has the same behavior as $abstr_i(\mathcal{F})$ in terms of control flow. Besides, the insertion position is computed according to the transitive $preset$ and $postset$ of $\mathcal{F}_i$ with respect to partner i. This conserves the order between the fragments and ensures the behavioral compatibility after the propagation of the INSERT operation.
- The DELETE pattern reduces the process models of the affected by the change. The reduction is done in a symmetric way on both sides: p and the partners affected. The deletion of an activity on one side results in the deletion of the corresponding $\overline{a}$ on the other side. Structural and behavioral compatibilities are kept. However, this might result in some issues, i.e., an activity might wait for a data that would never arrive or send a message that might not be used. The solution proposed in Section IV treats a set of use cases where the correlated interactions are updated or deleted accordingly. This does not affect structural and behavioral compatibility of the propagation.
- As described, the propagation of the REPLACE pattern results in three scenarios; insert, delete, or merge. The merge may reduce or augment the target models of the affected partners. Assume that the merge function γ is correct and idempotent and that it conserves the behavior of the merged fragments. We consider $\mathcal{F}_i$ and $\mathcal{F}'_i$ as the fragments to be merged. Then, the behavior of $\mathcal{F}_i$ is included in the merge result $\gamma(\mathcal{F}_i, \mathcal{F}'_i)$, which is also compatible with the behavior of p (changed process) since we merge the abstraction of the changed fragment (cf. Lemma 1).

The consistency between the public views and the private ones of the processes of the affected partners is checked if the negotiations succeed. Each affected partner must adapt its private process with respect to its modified public view. Using consistency rules, each partner can then check locally whether its local process model is consistent with its behavioral interface. More details about consistency checking and validation in choreographies can be found in [1], [8].

VI. Related Work

[19] proposes four transfer rules to deal with dynamic changes of distributed business processes. These rules use projection/protocol and lifecycle inheritance relations and check whether a changed process is a subclass of the original one. The suggested method only allows for changes that preserve inheritance transformation rules, i.e., changes having internal effects. Therefore, there is neither a need for change propagation nor new agreement on the global protocols since only inheritance-conforming changes are allowed. By contrast, our method supports changes that affect the external behavior of a process, and computes and propagates the respective changes to the affected partners according to a negotiation phase.

In [7], the authors consider change propagation in decentralized orchestrations where a centralized process model is split into several distributed partitions. They mainly propagate change operations made on the centralized model to the respective derived decentralized partitions. They use the decentralization function to compute the affected partitions and infer the respective changes to the private and public views. One organization controls the centralized and decentralized models. Hence, it is easier to exactly compute the affected regions and the changes to be applied. This is different from the current work since changes are made by one partner participating in the choreography and then propagated to the other partners. In this work, we consider fully distributed processes where no partner holds informations about another partner's private view (no centralized process model). Each partner can only see the public parts of the other partners. This leads to a negotiation phase between the affected partners and could have transitive structural and semantical effects. In contrast with the cited paper, in this work we take into consideration the semantic issues (transitivity). This problem is not considered when we propagate from a centralized to its decentralized partitions since the changes are made on the

centralized model and considered to be correct. An approach similar to [7] is also presented in [20].

In [4], the authors present DYCHOR, a framework addressing the challenge of change propagation in choreographies. Changes are classified into additive and subtractive, and may have variant or invariant impact on the interactions. DYCHOR uses annotated finite state automata to model choreographies and employs a set of operators to compute the changes to be propagated. In our work, we propose four change patterns which deal with more complex fragments instead of single activities only. This leads to new challenges concerning semantical or structural transitivity effects as well as negotiations with the partners affected by the change. We sketched several semantic transitive effects as well as the solutions to deal with them. The model adopted in this paper makes change propagation more easier since process models are structured and only changed regions are affected. Hence, there is no need for re-computing the whole public views of the processes of affected partners, instead only the affected regions are modified.

In [21], the authors address the problem of dynamic changes and particularly version management of process models. In the same context in [22], the authors propose an ontology based framework for decentralized workflow change management and define migration rules to adapt to changes in a dynamic way. In [23], the authors deal with change propagation between semantically overlapping process models for which elementary and complex correspondences have been identified. All these approaches are different but complementary to our work.

VII. Conclusion, Discussion, and Outlook

This paper provides algorithms for propagating process changes in collaborative scenarios involving multiple partners. To stay independent of a particular process specification language, RPST is used to define local and global choreography models. The proposed propagation algorithms consider typical process change patterns such as INSERT, DELETE, REPLACE, and UPDATE, and are evaluated based on their structural and behavioral compatibility.

Certain assumptions are made in this paper. First, the proposed algorithms consider the application of one change pattern at a time. However, in practical scenarios, several change patterns might be applied in a combined manner within a change transaction. To incorporate such complex changes, optimizations on the change transactions as suggested in [24] can be utilized to calculate the actual effects of the change transaction. Second, change propagation might become necessary in a transitive way, i.e., several partners might be affected. This can be handled by applying the change propagation procedure depicted in Fig. 4 in an iterative way. However, it must be considered whether the transitive propagation becomes cyclic. In this case, mechanisms such as upper bounds on the number of iterations of propagating changes including rollback mechanisms are conceivable.

Currently, we are integrating change propagation algorithms into our cloud-based process execution engine CPEE (cpee.org). We aim at testing and applying the proposed algorithms in case studies, for example, within the automotive domain.

References

[1] F. Casati, S. Ceri, B. Pernici, and G. Pozzi, "Workflow evolution," *Data and Knowledge Engineering*, vol. 24, no. 3, pp. 211–238, 1998.

[2] S. Rinderle, M. Reichert, and P. Dadam, "Correctness criteria for dynamic changes in workflow systems: a survey." *Data and Knowledge Engineering*, vol. 50, no. 1, pp. 9–34, 2004.

[3] M. Reichert and B. Weber, *Enabling Flexibility in Process-Aware Information Systems - Challenges, Methods, Technologies.* Springer, 2012.

[4] S. Rinderle, A. Wombacher, and M. Reichert, "Evolution of process choreographies in DYCHOR," in *CoopIS'06. LNCS*, 2006, pp. 273–290.

[5] S. Rinderle, B. Weber, M. Reichert, and W. Wild, "Integrating process learning and process evolution - a semantics based approach," in *Int'l Conference Business Process Management*, 2005, pp. 252–267.

[6] S. Rinderle, M. Reichert, and P. Dadam, "Flexible support of team processes by adaptive workflow systems." *Distributed and Parallel Databases*, vol. 16, no. 1, pp. 91–116, 2004.

[7] W. Fdhila, A. Baouab, K. Dahman, C. Godart, O. Perrin, and F. Charoy, "Change propagation in decentralized composite web services," in *CollaborateCom*, 2011, pp. 508–511.

[8] F. M. Besson, P. M. Leal, and F. Kon, "Towards verification and validation of choreographies," Department of Computer Science - University of So Paulo, technical research report, 2011.

[9] J. Vanhatalo, H. Vlzer, and J. Koehler, "The refined process structure tree," in *Business Process Management*, vol. 5240, 2008, pp. 100–115.

[10] B. Weber, M. Reichert, and S. Rinderle-Ma, "Change patterns and change support features - enhancing flexibility in process-aware information systems." *Data and Knowledge Engineering*, vol. 66, no. 3, pp. 438–466, 2008.

[11] G. Decker and M. Weske, "Behavioral consistency for B2B process integration," in *CAiSE*. Springer, 2007, pp. 81–95.

[12] A. Barros, M. Dumas, and P. Oaks, "Standards for web service choreography and orchestration: Status and perspectives," in *Business Process Management Workshops*. Springer. LNCS, 2006, vol. 3812, pp. 61–74.

[13] A. Polyvyanyy, L. Garcia-Banuelos, and M. Dumas, "Structuring acyclic process models," *Information Systems*, vol. 37, no. 6, pp. 518–538, 2012.

[14] R. Milner, M. Tofte, and D. Macqueen, *The Definition of Standard ML*. Cambridge, MA, USA: MIT Press, 1997.

[15] B. F. van Dongen, W. M. P. van der Aalst, and H. M. W. Verbeek, "Verification of EPCs: Using reduction rules and petri nets," in *CAiSE*. Springer, 2005, pp. 372–386.

[16] R. Dijkman, "Diagnosing Differences between Business Process Models," in *Proceedings of the 6th Int'l Conference on Business Process Management (BPM 2008)*. Springer, LNCS, 2008, pp. 261–277.

[17] M. L. Rosa, M. Dumas, R. Uba, and R. M. Dijkman, "Business process model merging : An approach to business process consolidation," *ACM Transactions on Software Engineering and Methodology*, 2012.

[18] F. Gottschalk, W. M. Aalst, and M. H. Jansen-Vullers, "Merging event-driven process chains," in *OTM '08*. Springer, 2008, pp. 418–426.

[19] W. M. P. van der Aalst and T. Basten, "Inheritance of workflows: an approach to tackling problems related to change," *Theor. Comput. Sci.*, vol. 270, no. 1-2, pp. 125–203, 2002.

[20] M. Reichert and T. Bauer, "Supporting ad-hoc changes in distributed workflow management systems." in *CoopIS'07*, vol. 4803. Springer, 2007, pp. 150–168.

[21] J. M. Küster, C. Gerth, and G. Engels, "Dynamic computation of change operations in version management of business process models," in *ECMFA'10*. Springer, 2010, pp. 201–216.

[22] V. Atluri and S. A. Chun, "Handling dynamic changes in decentralized workflow execution environments," in *DEXA'03*, 2003, pp. 813–825.

[23] M. Weidlich, J. Mendling, and M. Weske, "Propagating changes between aligned process models," *Journal of Systems and Software*, vol. 85, no. 8, pp. 1885 – 1898, 2012.

[24] S. Rinderle, M. Reichert, M. Jurisch, and U. Kreher, "On representing, purging, and utilizing change logs in process management systems," in *Business Process Management*, 2006, pp. 241–256.