Decomposition-based Verification of Global Compliance in Process Choreographies

Walid Fdhila
SBA-Research,
Vienna, Austria
wfdhila@sba-research.org

Stefanie Rinderle-Ma
Faculty of Computer Science
University of Vienna
Vienna, Austria
stefanie.rinderle-ma@univie.ac.at

David Knuplesch
Institute of Databases and Information Systems
Ulm University
Ulm, Germany
david.knuplesch@uni-ulm.de

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

***Abstract*—The verification of global compliance rules (GCR) in process choreographies (e.g., partner-spanning quality assurance in supply chains) is crucial and challenging due to the restricted visibility of the private processes of the collaborating partners. This paper provides a novel algorithm that decomposes global compliance rules into assertions that can be verified by the partners in a distributed way without revealing any private process details. The decomposition is based on transitivity properties of the underlying GCR specification. This work uses GCR based on antecedent and occurrence patterns and illustrates the transitivity properties based on their specification in first order predicate logic. It is formally shown that the original GCR can be reconstructed from the assertions, which ensures the viability of the approach. The algorithms are prototypically implemented and applied to several scenarios. The ability of checking global compliance constitutes a fundamental pillar of any approach implementing process choreographies with multiple partners.**

***Index Terms*—Distributed and cross-organizational business processes, Business rules and compliance management, Business process compliance, Global compliance rules, Rule decomposition**

I. Introduction

The arrival of technologies such as blockchain and Industry 4.0 has pushed the process-oriented collaboration between distributed business partners [25], realized by so called process choreographies. Figure 1 depicts a collaboration diagram of a supply chain scenario as a composition of the public models [1] of the partners *Manufacturer*, *Middleman*, *Special Carrier*, and *Supplier*. The private models of the partners, in turn, implement and possibly extend the behavior of the public models. **Private tasks**–depicted as dashed boxes in Fig. 1–are not visible to the partners, whereas **public tasks** can realize interaction (message exchange) between two partners. Although private tasks are usually hidden to other partners, restrictions over them might exist. In this case, no information about how and when the task is executed, or how it is connected to the other nodes of the corresponding private model is visible to the other partners. Usually, this happens when a collaborating partner imposes the execution of a specific task which must exist in its private process and comply with a given rule to another partner. The latter should then assure the existence of such task, and that it follows the imposed rule. Assume that the supply chain depicted in Fig. 1 is subject to a Global Compliance Rule (GCR), which stems from legal regulations and standards such as GDPR or Sarbanes Oxley. A GCR constrains actions of multiple partners and/or the interactions between them. Ensuring the compliance of process choreographies with a GCR is crucial and challenging [16] as a partner *"only has the visibility of the portion of the process under its direct control"* [26]. In Fig. 1, GCR C1 asks for a *safety check*, which is accomplished by a private task at the *Special Carrier.* To cope with this issue, *assertions* can be used. An assertion (A) is a commitment of a partner guaranteeing that its private/public process complies with the constraint [8]. In the example, the *Middleman* agrees to get the permission of the authority before ordering the sepcial transport (A1). *Special carrier* commits to perform a safety check before transporting the intermediate (A2). Combined, A1 and A2 enable checking GCR C1.

[26] tackles the problem of checking GCR on private tasks based on IoT-enabled artifacts. The approach presented in this paper aims to exploit assertions in order to stay independent of a particular technology such as IoT. The fundamental research question addressed in this work is as follows:

How to decompose a GCR into derived assertions that can be checked in a distributed way without violating the privacy of the involved process partners?

We tackle the question by providing a *decomposition algorithm* that breaks down the initial GCR into derived assertions. By combining the derived assertions, in turn, we can reconstruct the original GCR, i.e., the decomposition is lossless. In turn, the assertions obtained from the decomposition can be checked by the process partners in a distributed way without revealing any private details. The decomposition relies on

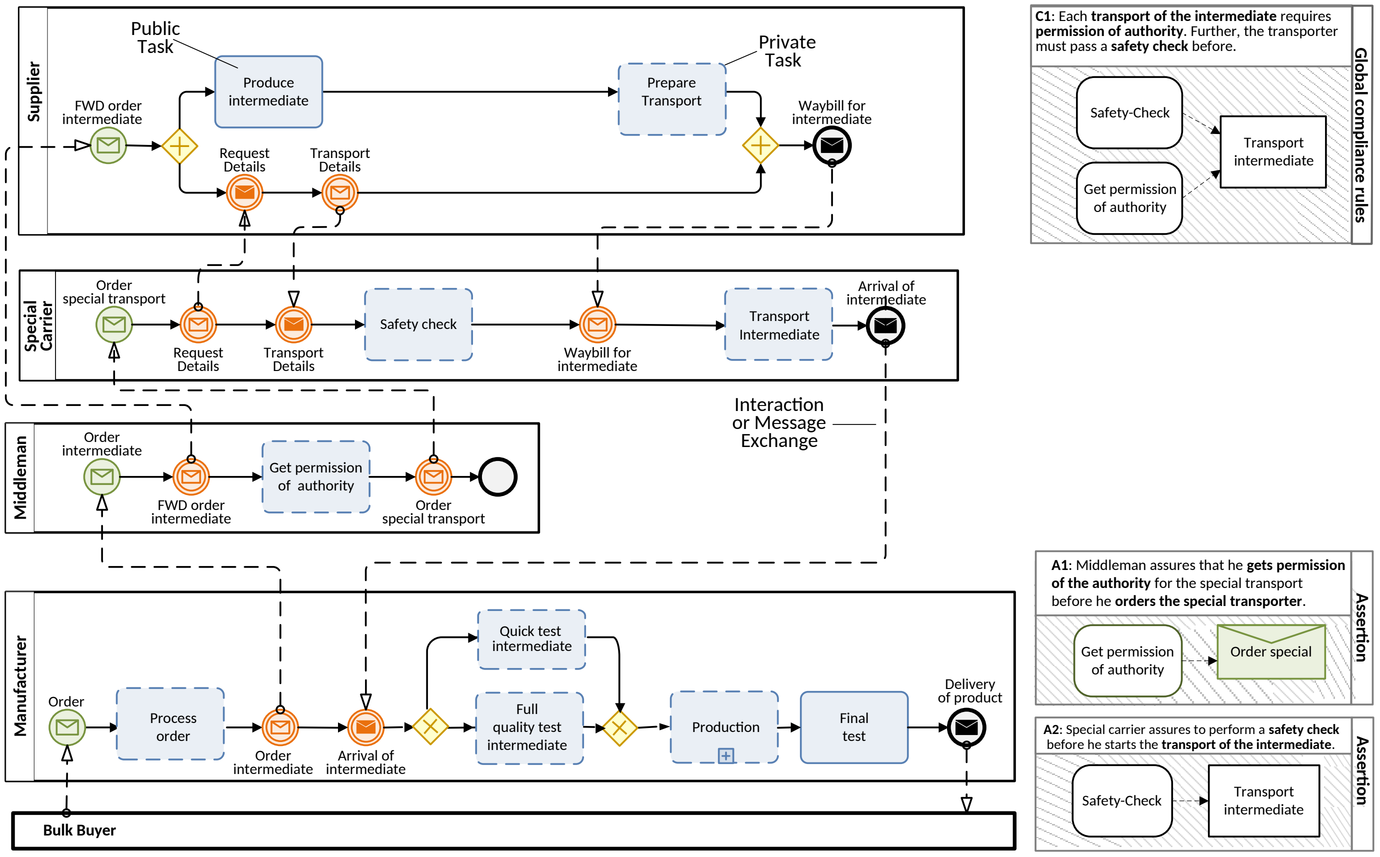

Fig. 1. Running example with four process partners: Supply chain (adapted from [8])

transitivity properties of the underlying GCR specification. In order to remain independent of a certain language and technology, we rely on simple rules that consist of antecedent and occurrence patterns. The transitivity properties are shown by the example of a translation to first order predicate logic. Similarly, for example, [29] presents declarative patterns based on Linear Temporal Logic (LTL). As part of the evaluation, the correctness of the decomposition algorithm is formally proven. Moreover, the algorithm is prototypically implemented and its application is demonstrated along basic choreography scenarios.

The remainder of the paper is structured as follows: Section II provides fundamentals. Section III-B deals with the decomposition algorithm and Sect. IV with the correctness of the algorithm. Sections V and III-A present details on the implementation and application of the algorithm. Section VI discusses related work and Section VII summarizes the paper.

II. Fundamentals and Transitivity Theorem

This section presents definitions and formal backgrounds for global compliance rules (GCR) over a process choreography y. A choreography includes three types of overlapping models: (i) a private model representing the executable process and including private activities as well as interactions with other partners, (ii) a public model (also called the interface of the process) that solely highlights the interactions of a given partner, and (iii) a choreography model providing a global view on the interactions between all partners [7].

The interactions of a single partner are described in a public model. The public model thus serves as public (restricted) view on the private model of the partner which *"describes the internal logic of a partner including its private activities and interactions"* [4]. In this paper, we assume that the choreography y is sound, i.e., *structural and behavioral compatibility* between public models and *consistency* between public and private models are given [7]. For a formal definition of process choreography, we refer the reader to Def. 3 in Appendix A.

We opt for expressing a GCR based on *antecedence* and *consequence patterns*. An antecedence pattern specifies when a GCR over y shall be triggered. In this situation, the desired behavior of y is described by related *consequence patterns*. Based on execution traces capturing all events related to the enactment of y, antecedence and consequence patterns are interpreted as follows: for each match of the antecedence pattern there must be at least one match of a consequence pattern. This simple representation of GCR abstracts from a certain language and technology, i.e., any language can be chosen that supports antecedence and consequence patterns, and the transitivity properties defined in this paper. Antecedence and consequence patterns are specified using *occurrence* and *absence nodes*, which either express the occurrence or absence of events related to the execution of activities or the exchange of messages in y. We use the following notation: $\boxed{A}$: Antecedence occurrence; $\xcancel{\boxed{A}}$: Antecedence absence; $\textcircled{A}$: Consequence occurrence; $\xcancel{\textcircled{A}}$: Consequence absence

In a previous work [6], multiple formal languages employed for business process compliance modelling and checking (e.g., linear temporal logic LTL, event calculus EC, extended compliance rule graph eCRG) were compared. It was proven that most of these languages can deal with most qualitative time patterns, and can therefore be used to model the compliance constructs addressed in this paper. Similar results were proven in [14]. In order to specify these constructs as well as transitivity properties required for the GCR decomposition, this work uses the eCRG formalism [21]. The latter offers a visual rule notation for expressing compliance rule graphs over choreographies and is based on first order predicate logic. Definition 1 specifies those parts of the eCRG formalism that are necessary for defining GCRs and their decompositions. In detail, Def. 1 (i) refers to activities and interactions in y, (ii) allows for exactly one antecedence occurrence pattern per rule, and (iii) assumes the rules to be structured as trees. As the decomposition focuses on the interactions between the partners and their control flows, (i) does not impose any restriction. Moreover, (iii) contributes to simplify the algorithm. For the sake of space and simplicity, we restrict the definition of GCR to one antecedence occurrence pattern (ii), but will extend this in future work.

Definition 1 (GCR structure): Given a process choreography y (cf. Def. 3 in Appendix A), let $\mathcal{A}$ be the set of public and private activities and $\mathcal{I}$ be the set of interactions in choreography model $\mathcal{G}$. Then: a GCR r is defined as tree $r = (N, \rho, \varphi, type, pattern, root)$ with

- N being the set of *nodes*,
- $\rho : N \rightarrow \{p_1 ... p_n\}$ returning the partner responsible for a node.
- $\varphi : N \times N \rightarrow \{ \dashrightarrow, \longrightarrow \}$ returning the sequence flow connector of two nodes, i.e., consequence sequence and antecedence sequence connectors respectively.
- $type : N \rightarrow \mathcal{A} \cup \mathcal{I}$ mapping each node to an *activity* or an *interaction*, i.e., message exchange.
- $pattern : N \rightarrow \{ \boxed{A}, \xcancel{\boxed{A}}, \textcircled{C}, \xcancel{\textcircled{C}} \}$
- $root \in N$ being the root node with pattern(root) = $\boxed{A}$
- $\forall n \in N \setminus \{root\} : pattern(n) \neq \boxed{A}$
- $\forall n \in N$ with pattern(n) $\in \{\xcancel{\boxed{A}}, \xcancel{\textcircled{C}}\}$: n is leaf node

We illustrate the translation of a GCR into a First Order Logic (FOL) expression using basic equivalences as in Def. 2 following below. As shown in the following, equivalences are also required for ensuring correctness of the decomposition. Additionally, throughout the paper we consider the consequence sequence connector. This can be easily extended to antecedence sequence connectors.

Definition 2 (Basic Equivalences): Based on [22] the following equivalences hold by definition. Predicate $x(t, ty)$ describes that at the point in time t an activity (message) of type ty was executed (i.e., sent or received).

- $\boxed{A} \dashrightarrow \textcircled{B} :\Leftrightarrow \forall a : \big(x(a,A) \rightarrow \big(\exists b : (x(b,B) \wedge a < b)\big) \big)$
  $\Leftrightarrow \forall a \exists b : \big(x(a,A) \rightarrow (x(b,B) \wedge a < b)\big)$
- $\textcircled{A} \dashrightarrow \boxed{B} :\Leftrightarrow \forall b : \big(x(b,B) \rightarrow \big(\exists a : (x(a,A) \wedge a < b)\big) \big)$
  $\Leftrightarrow \forall b \exists a : \big(x(b,B) \rightarrow (x(a,A) \wedge a < b)\big)$
- $\boxed{A} \dashrightarrow \xcancel{\textcircled{B}} :\Leftrightarrow \forall a :$
  $\big(x(a,A) \rightarrow \big(\nexists b : (x(b,B) \wedge a < b)\big) \big)$
  $\Leftrightarrow \forall a, b : \big(x(a,A) \rightarrow \neg(x(b,B) \wedge a < b)\big)$
  $\Leftrightarrow \forall a, b : \big(x(a,A) \rightarrow (\neg x(b,B) \vee b \leq a)\big)$
- $\xcancel{\textcircled{A}} \dashrightarrow \boxed{B} :\Leftrightarrow \forall b :$
  $\big(x(b,B) \rightarrow \big(\nexists a : (x(a,A) \wedge a < b)\big) \big)$
  $\Leftrightarrow \forall b, a : \big(x(b,B) \rightarrow \neg(x(a,A) \wedge \neg a < b)\big)$
  $\Leftrightarrow \forall a, b : \big(x(a,A) \rightarrow (\neg x(b,B) \vee b \leq a)\big)$

For example, GCR $\boxed{\text{Production}} \dashrightarrow \textcircled{\text{Final test}}$ is translated into: $\forall a : \big(x(a, \text{Production}) \rightarrow \exists b : x(b, \text{Final test}) \wedge a < b\big)$. Relation $<$ expresses a temporal precedence between a and b. The decomposition algorithm presented in Sect. III-B exploits the transitivities for GRC as in Theorem 1. Specifically, by combining transitive relations, where each relation can be checked locally by a single partner, it becomes possible to reconstruct the original GCR behavior. Theorem 1 ensures that the behavior of the derived assertions reproduces the behavior of the GCR, but not vice versa.

Theorem 1 (Transitivities):
Let A, B, and C be three activity or message types. Then:

a. The rightwards transitivity holds:
   $\boxed{A} \dashrightarrow \textcircled{B} \wedge \boxed{B} \dashrightarrow \textcircled{C} \Rightarrow \boxed{A} \dashrightarrow \textcircled{C}$
b. The leftwards transitivity holds:
   $\textcircled{A} \dashrightarrow \boxed{B} \wedge \textcircled{B} \dashrightarrow \boxed{C} \Rightarrow \textcircled{A} \dashrightarrow \boxed{C}$
c. The rightwards zig zag transitivity holds for the consequence absence:
   $\textcircled{B} \dashrightarrow \boxed{A} \wedge \boxed{B} \dashrightarrow \xcancel{\textcircled{C}} \Rightarrow \boxed{A} \dashrightarrow \xcancel{\textcircled{C}}$
d. The leftwards transitivity holds for the consequence absence:
   $\boxed{A} \dashrightarrow \textcircled{B} \wedge \xcancel{\textcircled{C}} \dashrightarrow \boxed{B} \Rightarrow \xcancel{\textcircled{C}} \dashrightarrow \boxed{A}$

In the following, the correctness of Theorem 1 is proven applying Def. 2.

Proof 1 (Rightwards Transitivity): Let A, B, C be three activities or interactions. Then $\boxed{A} \dashrightarrow \textcircled{B} \wedge \boxed{B} \dashrightarrow \textcircled{C}$
$:\Leftrightarrow \forall a \exists b : \Big(x(a,A) \rightarrow (x(b,B) \wedge a < b)\Big) \wedge \forall b \exists c : \Big(x(b,B) \rightarrow (x(c,C) \wedge b < c)\Big)$
$\Leftrightarrow \forall a \Big(\exists b : \big(x(a,A) \rightarrow (x(b,B) \wedge a < b)\big) \wedge \forall b : \exists c : \big(x(b,B) \rightarrow (x(c,C) \wedge b < c)\big)\Big)$
$\Rightarrow \forall a \exists b, c : \Big(\big(x(a,A) \rightarrow (x(b,B) \wedge a < b)\big) \wedge \big(x(b,B) \rightarrow (x(c,C) \wedge b < c)\big)\Big)$
$\Rightarrow \forall a \Big(\exists b, c : \Big(\big(x(a,A) \rightarrow x(b,B) \rightarrow (x(c,C) \wedge a < b < c)\big)\Big)\Big)$
$\Rightarrow \forall a \exists c : \Big(x(a,A) \rightarrow (x(c,C) \wedge a < c)\Big) \Rightarrow \boxed{A} \dashrightarrow \textcircled{C}$
q.e.d.

Leftwards transitivity can be proven similarly by replacing '$<$' with '$>$'.

Corollary 1: Let A, B, C, and D be activities or interactions. Then

$\boxed{A} \dashrightarrow \textcircled{B} \wedge \boxed{B} \dashrightarrow \textcircled{C} \wedge \boxed{C} \dashrightarrow \textcircled{D} \Rightarrow \boxed{A} \dashrightarrow \textcircled{C} \wedge \boxed{C} \dashrightarrow \textcircled{D}$
$\Rightarrow \boxed{A} \dashrightarrow \textcircled{D}$

Proof 2 (Rightwards Zig Zag Transitivity of Absence): Let A, B, C be activities or interactions. Then:

$\textcircled{B} \dashrightarrow \boxed{A} \wedge \boxed{B} \dashrightarrow \cancel{\textcircled{C}}$

$:\Leftrightarrow \quad \forall a \exists b : \Big(x(a, A) \rightarrow (x(b, B) \wedge b < a)\Big) \wedge \forall b, c : \Big(x(b, B) \rightarrow (\neg x(c, C) \vee c \leq b)\Big)$

$\Leftrightarrow \quad \forall a \Big(\exists b : \big(x(a, A) \rightarrow (x(b, B) \wedge b < a)\big) \wedge \forall b, c : \big(x(b, B) \rightarrow (\neg x(c, C) \vee c \leq b)\big)\Big)$

$\Rightarrow \quad \forall a \exists b \forall c : \Big(\big(x(a, A) \rightarrow (x(b, B) \wedge b < a)\big) \wedge \big(x(b, B) \rightarrow (x(c, C) \rightarrow c \leq b)\big)\Big)$

$\Rightarrow \quad \forall a \exists b \forall c : \Big(\big(x(a, A) \rightarrow x(b, B) \rightarrow (x(c, C) \rightarrow c \leq b < a)\big)\Big)$

$\Rightarrow \quad \forall a \forall c : \Big(x(a, A) \rightarrow (x(c, C) \rightarrow c \leq a)\Big)$

$\Rightarrow \quad \forall a, c : \Big(x(a, A) \rightarrow (\neg x(c, C) \vee c \leq a)\Big) \Rightarrow \boxed{A} \dashrightarrow \cancel{\textcircled{C}}$

q.e.d.

Leftwards *zig zag* transitivity of absence can be proven similarly by replacing '$<$' with '$>$' and '$\leq$' with '$\geq$'. The transitivity of other constructs such as ***"chaining" or "requires transitivity"*** can be found in Appendix B.

III. Decomposing Global Compliance Rules

At design time, checking a GCR that solely refers to interactions and/or public activities can be achieved by applying contemporary compliance checking techniques (cf. [13]) either on the choreography model or the public models of the involved partners. By contrast, if a GCR refers to private activities of different partners, it becomes impossible to check its correctness as partners cannot view the private process parts of the other partners and, therefore, cannot identify the dependencies between the private activities involved in the GCR. To cope with this, we suggest decomposing the GCR into a set of assertions of which each can be checked locally by the corresponding partner. The decomposed rules then reproduce the behavior of the original GCR.

Decomposition in compliance checking has been exploited by [32], but only at the process model level in order to achieve performance gains for the compliance checks. The paper at hand, first of all, proposes to decompose the global compliance rule to distribute the compliance checks to the partners for maintaining the confidentiality of their private tasks. We start with illustrating the idea and the challenges of GCR decomposition based on different scenarios. Then the decomposition algorithm is introduced and explained.

A. GCR Decomposition Scenarios and Discussion

In the following, we use a set of scenarios (cf. Figs. 2 and 3) to illustrate the GCR decomposition process. For the sake of readability, in these scenarios we assume a simple compliance rule involving two tasks of two different partners p_1 and p_2. In Scenarios 1 – 3, we assume rule $\boxed{A} \dashrightarrow \textcircled{B}$, where the execution of task A shall be followed by the one of task B. For simplicity, we use labels A, B, $m1$, $m2$, p_1 and p_2 instead of the actual labels taken from the example of Fig. 1.

- In Scenario 1 there exists a message exchange $m1$ between p_1 and p_2 that fulfills conditions $\boxed{A} \dashrightarrow \textcircled{m1}$ and $\boxed{m1} \dashrightarrow \textcircled{B}$. The behavioral and structural compatibility (cf. Sect. II) between the partner processes ensures that message $m1$ sent by p_1 shall be correctly received by p_2.
- In Scenario 2 we assume that there is no direct interaction between p_1 and p_2, but there exists a transitive interaction through partner p_3. Partner p_1 interacts with p_3 through $m1$ and p_3 interacts with p_2 through $m2$. In this scenario, the transitive relations $\boxed{A} \dashrightarrow \textcircled{m1}$, $\boxed{m1} \dashrightarrow \textcircled{m2}$, and $\boxed{m2} \dashrightarrow \textcircled{B}$ reproduce the behavior of $\boxed{A} \dashrightarrow \textcircled{B}$. Consequently, partner p_3, which is initially not involved in the GCR becomes involved in the derived assertions. We call p_3 an *intermediary partner.*
- In Scenario 3 we assume that, even though there are interactions between p_1 and p_2, no direct or transitive interactions reproduce the behavior of $\boxed{A} \dashrightarrow \textcircled{B}$. As such, additional message exchanges become necessary to inform about the execution state of activities involved in the GCR. Message exchanges can be either synchronous or asynchronous. Asynchronous message exchange only allows for reactive GCR checking and, hence, for detecting violations after their occurrence. Synchronous message exchange is proactive as it enforces the GCR with new restrictions to the process models, e.g., the execution of B is enabled only after receiving the synchronization message (i.e. whether A is or is not executed). Partner p_1 shall also inform p_2 in case A is not executed, as this does not prevent B from being executed according to the GCR.

Rightwards transitivity (cf. Theorem 1.a) directly ensures the correctness of Scenarios 1 and 3 from Figure 4. It should be clear that the correctness of Scenario 2 can be directly derived from Corollary 1 (cf. Sect. II).

For the scenarios depicted in Figure 3, we assume different compliance rules, where the antecedent presence $\boxed{B}$ implies the later consequence absence $\cancel{\textcircled{A}}$ in Scenario 4; i.e., $\boxed{B} \dashrightarrow \cancel{\textcircled{A}}$, and antecedence presence $\boxed{B}$ requires the consequence presence $\textcircled{A}$ before, in Scenario 5, i.e., $\textcircled{A} \dashrightarrow \boxed{B}$.

- In Scenario 4 (includes an adaptation of the example of Fig. 1), for partner p_1, the execution of A automatically implies non-execution of $m1$ before and vice versa (due to the used XOR pattern). Consequently, it ensures rule $\boxed{m1} \dashrightarrow \cancel{\textcircled{A}}$. For partner p_2, the execution of B is always preceded by the one of $m1$, which implies $\textcircled{m1} \dashrightarrow \boxed{B}$. The composition of the two rules $\boxed{m1} \dashrightarrow \cancel{\textcircled{A}}$ and $\textcircled{m1} \dashrightarrow \boxed{B}$, in turn, reproduces the behavior of $\boxed{B} \dashrightarrow \cancel{\textcircled{A}}$. Similarly to Scenario 2, this dependency can become more complex and transitive. This example also

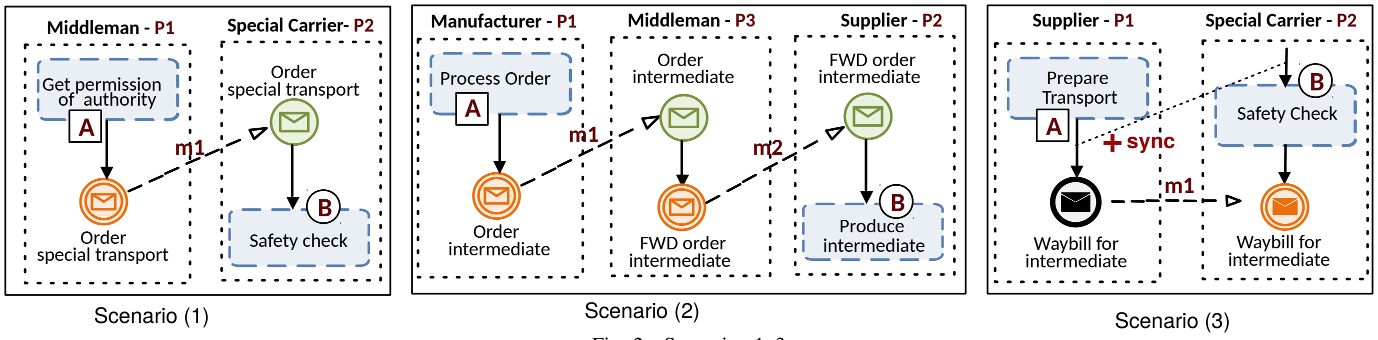

Fig. 2. Scenarios 1–3

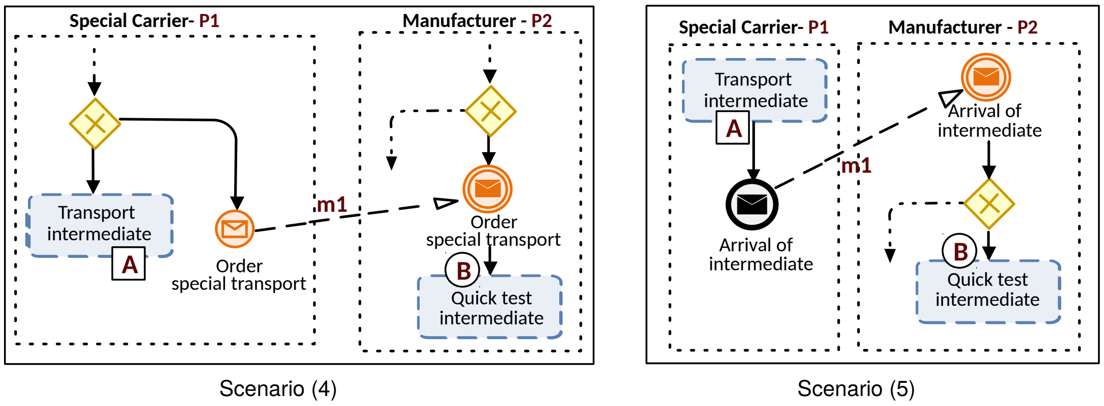

Fig. 3. Scenarios 4–5

illustrates $\boxed{A} \dashrightarrow \cancel{\textcircled{B}}$ and $\cancel{\textcircled{A}} \dashrightarrow \boxed{B}$ in a similar way.

- In Scenario 5, if B is executed, A should have been executed before. However, the execution of A does not imply that B is always executed. Regarding p_1, the execution of A implies the one of $m1$ and vice versa; i.e., $\boxed{A} \dashrightarrow \textcircled{m1}$ and $\textcircled{A} \dashrightarrow \boxed{m1}$. Regarding p_2, the execution of B always implies the one of $m1$ before, but not the other way around; i.e. $\textcircled{m1} \dashrightarrow \boxed{B}$. Since both assertions refer to the same message exchange $m1$ between p_1 and p_2, in combination with each other, the two assertions $\textcircled{A} \dashrightarrow \boxed{m1}$ and $\textcircled{m1} \dashrightarrow \boxed{B}$ can reproduce the behavior of $\textcircled{A} \dashrightarrow \boxed{B}$

The correctness of Scenario 5 concludes from the rightwards transitivity (cf. Theorem 1.b), whereas Scenario 4 relies on the zig zag transitivity of the absence (cf. Theorem 1.c). Figure 4 summarizes the decomposition results of the sketched scenarios. The latter are used to illustrate selected use cases. The decomposition process is not limited to these scenarios and, as aforementioned, the decomposition cannot always be automated, but might require manual interaction and processing. Altogether, the decomposition eases global compliance rule checking, where each partner checks its corresponding derived assertions locally. A GCR is rechecked only if at least one of the derived assertions is not evaluated to true. Note that this does not necessarily imply that the GCR is violated.

B. GCR Decomposition Algorithm

This section focuses on the decomposition algorithm and explains the steps to derive assertions using transitivities. Algorithm 1 realizes GCR decomposition as set out in Def. 1. It assumes that each node of the GCR is assigned to one partner being responsible for it. Further on, we assume the input GCR to be consistent and satisfiable (for dealing with unsatisfiable and inconsistent rules we refer to [5]). In the following, we first explain Algorithm 1 step by step and then illustrate the entire algorithm along Example 1 (see below).

Starting from the $\boxed{A}$ node (cf. Def. 1), Algorithm 1 walks outwards through all other nodes of the GCR. Thereby, the visited parts are copied and become assertions. Wherever the algorithm walks over a connector between two nodes n and s, which are assigned to different partners $\rho(n)$ and $\rho(s)$, the GCR is split at this position as this dependency cannot be evaluated by a single partner. Next, the algorithm tries to replicate the connector where the GCR was split through (transitive) message exchanges between both affected partners by applying the transitive relationships from Section II. Therefore, sets $n\bullet$, $\bullet s$, and Θ are calculated. Depending on the pattern of s (cf. Def. 1), $n\bullet$ and $\bullet s$ contain the messages succeeding or preceding n and s. Note that these calculations have to be accomplished in a decentralized manner by $\rho(n)$ and $\rho(s)$ themselves as n and s may be private tasks. Next, Θ combines those messages of $n\bullet$ and $\bullet s$ that can be combined. If s is a $\textcircled{C}$ node (i.e., s must follow n), Θ contains message tuples (m_1, m_2) that ensure that n is always followed by

m_1, m_1 by m_2 (unless $m_1 = m_2$), and m_2 by s. Any pair $(m_1, m_2) \in \Theta$ can then be used to complement the created assertions, i.e., m_1 becomes a placeholder for s within the assertion of $\rho(n)$, whereas m_2 replaces n for $\rho(s)$.

Regarding $\cancel{\textcircled{B}}$ nodes (i.e., s must not follow n), all pairs of messages $(m_1, m_2) \in \Theta$ ensure that n is preceded by m_1 in any case and m_1 is preceded by m_2 (unless $m_1 = m_2$), whereas s never follows m_2. Finally, for $\cancel{\boxed{A}}$ nodes an occurrence of s after n allows ignoring the rule. Hence, $\cancel{\boxed{A}}$ nodes result in pairs (m_1, m_2) such that m_1 may only occur after n and m_2 may only occur after m_1 (unless $m_1 = m_2$). However, there should be at least one case in which m_2 is followed by s (i.e., s is not always preceded by m_2).

Finally, all assertions of the same partner, which depend on the same $\boxed{A}$ message, are merged to reduce the number of assertions. Remaining assertions without consequences are removed as they result from the processing of $\cancel{\boxed{A}}$ nodes, but have not been merged in the previous step. Remember that ignoring $\cancel{\boxed{A}}$ nodes is allowed as this makes rules even more strict.

Example 1: Let us apply Alg. 1 to GCR C1 from the running example in Fig. 1. Let the responsibilities be ρ(*Safety Check*) = *Special Carrier*, ρ(*Get permission of authority*) = *Middleman*, and ρ(*Transport intermediate*) = *Special Carrier*. After assigning responsibilities (cf. labels above the nodes), Algorithm 1 starts with $\boxed{A}$ node *Transport intermediate* and creates a new assertion for the *Special Carrier* who is responsible for this task. Then the *Safety check* is discovered and added to the assertion, since it belongs to the same partner. In turn, another partner (i.e., *Middleman*) is responsible for task *Get permission of authority*. Hence, the algorithm cuts the respective connector and creates a new assertion for the respective partner. Next, *Special Carrier* and *Middleman* determine $n\bullet$ and $\bullet s$ with $n\bullet = \{\mathit{Waybill}, \mathit{T.\ Details}, \mathit{Req.\ Details}, \mathit{Order\ ST}\}$ and $\bullet s = \{\mathit{Order\ ST}\}$ to calculate those message pairs $\Theta = \{(\mathit{Waybill}, \mathit{Order\ ST}), \ldots, (\mathit{Order\ ST}, \mathit{Order\ ST})\}$ that can be used to transitively replicate the connector where the GCR was split. Finally, the algorithm places the selected messages into both assertions in such a way that the correctness of the original rule is preserved through the (leftwards) transitivity of eCRGs. Note that the *Special Carrier* could use message *Waybill* instead since $(\mathit{Waybill}, \mathit{Order\ ST}) \in \Theta$ holds.

For the same GCR, it is possible to infer several decompositions, depending on which interactions are used to find a transitive control flow relationship between the nodes of the GCR. It is also possible that no direct link can be identified between two partners' GCR nodes (i.e., there is no interaction between those two partners). As such, interactions with *intermediary partners* can be used to find an indirect link (i.e., transitive interactions). As aforementioned, if no transitivity is identified between the GCR nodes of two partners (even not through intermediary partners), it becomes necessary to exchange additional execution data between the partners involved in the GCR, by, for example, adding sync messages. Sync messages are specific type of messages communicated between partners to inform about the state of a given task (e.g., terminated, started, not executed). Although sync messages are not preferred as they expose private data about the exact execution time of a private task, they become necessary when the GCR cannot be decomposed into assertions, i.e., no transitive relations can be identified.

IV. Correctness and Complexity of GCR Decomposition

At first, this section formally proves the correctness of the GCR decomposition into assertions (cf. Alg. 1). In detail, it is shown that we can reconstruct the original GCR from the assertions, i.e., the decomposition is lossless[1].

Let $\Delta = (N, \rho, \varphi, \mathit{type}, \mathit{pattern}, \mathit{root})$ be a GCR (cf. Def. 1). For Δ, a partner p might be responsible (cf. Alg. 1) for multiple nodes within N, e.g., if multiple nodes of N belong to the process of p. In this scenario, the decomposition of Δ might result in multiple sub GCRc (subgraphs) to be checked by p, in particular if the nodes of p are dispersed over Δ and are not directly connected. Let $\{\Delta_i\}_p$ be the set of sub GCRs issued from the decomposition and involving partner p.

Let's consider two nodes n and $n' \in N$, the corresponding connector $\varphi(n, n\prime)$, and $\Psi(n, n\prime)$ = (pattern(n), pattern($n\prime$),$\varphi(n, n\prime)$) as the corresponding assertion. Now assume there exists a set of message interaction nodes $\{m_k\}_{k=1}^{k=h}$, such as $\Psi(n, m_1) \equiv \Psi(m_{k-1}, m_k)_{k=2}^{k=h} \equiv \Psi(m_h, n') \equiv$

[1] **Note:** For the sake of simplicity, the notation $\boxed{A} \dashrightarrow \textcircled{B} :\Leftrightarrow \forall a : (\ x(a, A) \rightarrow (\exists b : (x(b, B) \wedge a < b)$ has been simplified to: $\boxed{A} \dashrightarrow \textcircled{B} :\Leftrightarrow \forall a \rightarrow \exists b(\ \wedge a < b)$

Scenario	Global Compliance Rule (GCR)	Derived Assertions (AR)	Scenario	Global Compliance Rule (GCR)	Derived Assertions (AR)
(1)	$\boxed{A, p_1} \dashrightarrow (B, p_2)$	$\boxed{A, p_1} \dashrightarrow (m1, p_1)$ $\boxed{m1, p_2} \dashrightarrow (B, p_2)$	(2)	$\boxed{A, p_1} \dashrightarrow (B, p_2)$	$\boxed{A, p_1} \dashrightarrow (m1, p_1)$ $\boxed{m1, p_3} \dashrightarrow (m2, p_3)$ $\boxed{m2, p_2} \dashrightarrow (B, p_2)$
(3)	$\boxed{A, p_1} \dashrightarrow (B, p_2)$	$\boxed{A, p_1} \dashrightarrow (\mathit{sync}, p_1)$ $\boxed{\mathit{sync}, p_2} \dashrightarrow (B, p_2)$	(4)	$\boxed{B, p_2} \dashrightarrow \cancel{(A, p_1)}$	$\boxed{m1, p_1} \dashrightarrow \cancel{(A, p_1)}$ $(m1, p_2) \dashrightarrow \boxed{B, p_2}$
(5)	$(A, p_1) \dashrightarrow \boxed{B, p_2}$	$(A, p_1) \dashrightarrow \boxed{m1, p_1}$ $(m1, p_2) \dashrightarrow \boxed{B, p_2}$			

Fig. 4. GCR Decomposition Results

Algorithm 1: GCR decomposition DECOMPOSE(gcr)

```
• Global compliance rule $gcr = (N, \rho, \varphi, type, pattern, root)$
• Choreography model $y$, and $\mathcal{M}$ as the set of all partners' message nodes.
• We assume that $\rho$ also returns the partner private model of a node $n$.
select the only $a \in N$ with $pattern(a) = \boxed{A}$
initialize queue $Q \leftarrow \{a\}$
create (incomplete) Assertion $A_a \leftarrow$ "$\boxed{a}$" for the partner associated with $\rho(a)$
foreach ($n \leftarrow removeHead(Q)$) do
    foreach ($s \in N$ with $\varphi(n, s) \neq \emptyset$) do
        $Q \leftarrow Q \cup \{s\}$
        if ($\rho(n) = \rho(s)$) then
            //n and s involve the same partner
            initialize $A_s \leftarrow @A_n$ as reference on $A_n$
            if ($pattern(s) = \xcancel{\boxed{A}}$) then extend $A_s$ with "$\dashrightarrow \xcancel{\boxed{s}}$"
            if ($pattern(s) = \textcircled{C}$) then extend $A_s$ with "$\dashrightarrow \textcircled{s}$"
            if ($pattern(s) = \xcancel{\textcircled{C}}$) then extend $A_s$ with "$\dashrightarrow \xcancel{\textcircled{s}}$"
        else
            //n and s involve different partners $p_i$, $p_j$
            if ($pattern(s) = \textcircled{C}$) then
                $n\bullet \leftarrow \{m \in \rho(n) | m \in \mathcal{M}, \rho(n) \models \boxed{n} \dashrightarrow \textcircled{m}\}$
                $\bullet s \leftarrow \{m \in \rho(s) | m \in \mathcal{M}, \rho(s) \models \boxed{m} \dashrightarrow \textcircled{s}\}$
                $\Theta \leftarrow \{(m_n, m_s) \in (n\bullet \times \bullet s) \mid \gamma \models \boxed{m_n} \dashrightarrow \textcircled{m_s}\}$

            if ($pattern(s) = \xcancel{\textcircled{C}}$) then
                $n\bullet \leftarrow \{m \in \rho(n) | m \in \mathcal{M}, \rho(n) \models \textcircled{m} \dashrightarrow \boxed{n}\}$
                $\bullet s \leftarrow \{m \in \rho(n) | m \in \mathcal{M}, \rho(n) \models \boxed{m} \dashrightarrow \xcancel{\textcircled{s}}\}$
                $\Theta \leftarrow \{(m_n, m_s) \in (n\bullet \times \bullet s) \mid \gamma \models \textcircled{m_s} \dashrightarrow \boxed{m_n}\}$

            if ($pattern(s) = \xcancel{\boxed{A}}$)} then
                $n\bullet \leftarrow \{m \in \rho(n) | m \in \mathcal{M}, \rho(n) \models \textcircled{n} \dashrightarrow \boxed{m}\}$
                $\bullet s \leftarrow \{m \in \rho(s) | m \in \mathcal{M}, \rho(s) \not\models \boxed{m} \dashrightarrow \xcancel{\textcircled{s}}\}$
                $\Theta \leftarrow \{(m_n, m_s) \in (n\bullet \times \bullet s) \mid \gamma \models \textcircled{m_n} \dashrightarrow \boxed{m_s}\}$

            if ($(\Theta \cup (n\bullet \cap \bullet s) = \{\emptyset\})$) then
                //No implicit dependency between n and s
                add $sync$ message between $n$ and $s$
                update models $p_1, \ldots, p_n$, and $\gamma$
                recalculate $n\bullet$, $\bullet s$, and $\Theta$
            else
                //implicit dependency m between n and s exists
                select $(m_n, m_s) \in \Theta \cup (n\bullet \cap \bullet s)^2$
                if ($pattern(s) = \textcircled{C}$) then
                    extend $A_n$ with "$\dashrightarrow \textcircled{m_n}$"
                    create Assertion $A_s \leftarrow$ "$\boxed{m_s} \dashrightarrow \textcircled{s}$" for $\rho(s)$

                if ($pattern(s) = \xcancel{\textcircled{C}}$) then
                    extend $A_s$ with "$\textcircled{m_n} \dashrightarrow$ "
                    create Assertion $A_s \leftarrow$ "$\boxed{m_s} \dashrightarrow \xcancel{\textcircled{s}}$" for $\rho(s)$

                if ($pattern(s) = \xcancel{\boxed{A}}$) then
                    create Assertion $A_s \leftarrow$ "$\boxed{m_s} \rightarrow \xcancel{\boxed{s}}$" for $\rho(s)$

    foreach (($s \in N$ with $\varphi(n, s) \neq \emptyset$)) do
        //same as for each $(n, s) \in C$ above
        //but with flipped directions
    foreach (partner $i$) do
        foreach (($A_j$, $A_k$) of partner $i$) do
            if ($A_j$, $A_k$ have the same $\boxed{A}$ pattern) then
                merge $A_j$ and $A_k$ based on the $\boxed{A}$ pattern

    foreach (Assertion $A$) do
        if ($A$ has empty $\textcircled{C}$ and $\xcancel{\textcircled{C}}$ patterns) then remove $A$
```

$\Psi(n, n\prime)$, where $\Psi(n, m_1) \equiv \Psi(n, m_1)$ if and only if pattern(a) = pattern(c) and pattern (b)=pattern(d) and $\varphi(a, b) = \varphi(c, d)$. Then according to Theorem 1 it holds:

$$\Psi(n, m_1)[\bigwedge_{k=2}^{k=h} \Psi(m_{k-1}, m_k)] \wedge \Psi(m_h, n') \implies \Psi(n, n\prime)$$

This implies that the first assertion can be decomposed into a conjunction of finite sets of transitive relations. If there exists a message interaction m that transitively ensures the original relation between n_i and n_j, $\Psi(n_i, n_j)$ is decomposed into $\Psi(n_i, send(m)) \wedge \Psi(send(m), receive(m)) \wedge \Psi(receive(m), n_j)$. For the sake of simplicity, we directly use $\Psi(n_i, m) \wedge \Psi(m, n_j)$. Doing so is possible due to the compatibility property in process collaborations that ensures that each *send* task of one partner is necessarily connected to a *receive* task of another one. Note that a node $n \in N$ is not always solely connected to one single node, but may have multiple connections defining its semantics. For example, regarding GCR $C1$ (cf. Fig. 1), *Transport Intermediate* has two incoming connectors. We call such a node an *inner node*. In the following, we are interested in proving that the semantics of such nodes are preserved after decomposition. For the sake of readability, we consider a GCR composed of one single *inner node* of type $\boxed{A}$. However, the proof can be extended to multiple composite nodes and different node types (e.g., $\textcircled{C}$ $\xcancel{\boxed{A}}$), following the same logic. We define $\bullet n$ and $n\bullet$ as the set of all nodes preceding and succeeding n respectively. Consider a composite node n of type $\boxed{A}$, which is preceded by two sets of antecedence and consequence occurrence nodes (i.e., $\bullet A$ and $\bullet C$ respectively), and followed by two other sets of antecedence and consequence occurrence nodes (i.e., $A\bullet$ and $C\bullet$ respectively). Assume $\bullet A = \{a_i\}, A\bullet = \{b_j\}, \bullet C = \{c_k\}$, and $C\bullet = \{d_l\}$, then the generic FOL formula for such a node is given as follows [22]:
$\forall a_i, \forall b_j, \forall n : \bigwedge_i (a_i < n) \bigwedge_j (n < b_j) \rightarrow \exists c_k, \exists d_l : \bigwedge_k (c_k < n) \bigwedge_l (n < d_l)$
Using Theorem 1 each relation of type n precedes n' $(n < n')$; if there exists a message interaction m such as $n < m$ and $m < n'$, then:

$\forall a_i, \exists m_i, \forall b_j, \exists m_j, \forall n : \bigwedge_i (a_i < m_i) \bigwedge_i (m_i < n) \bigwedge_j (n < m_j) \bigwedge_j (m_j < n) \rightarrow$
$\exists c_k, m_k, \exists d_l, m_l : \bigwedge_k (c_k < m_k) \bigwedge_k (m_k < n) \bigwedge_l (n < m_l) \bigwedge_l (m_l < n)$

$$\Leftarrow$$

$\bigwedge_i (a_i \rightarrow m_i)$
$\bigwedge_j (m_j \rightarrow b_j)$
$[\forall m_i, \forall m_j, \forall n : \bigwedge_j (m_i < n) \bigwedge_j (n < m_j)$
$\rightarrow \exists m_k, \exists m_l : \bigwedge_k (m_k < n) \bigwedge_l (n < m_l) \exists c_k, \exists d_l : \bigwedge_k (c_k < m_k) \bigwedge_l (m_l < n)]$

$$\Leftarrow$$

$\bigwedge_i (a_i \rightarrow m_i)$
$\bigwedge_j (m_j \rightarrow b_j)$
$[\forall m_i, \forall m_j, \forall n : \bigwedge_j (m_i < n) \bigwedge_j (n < m_j)$
$\rightarrow \exists m_k, \exists m_l : \bigwedge_k (m_k < n) \bigwedge_l (n < m_l)\ \forall c_k, \forall d_l : \bigwedge_k (c_k < m_k) \bigwedge_l (m_l < n)]$

$$\Leftarrow$$

$\bigwedge_i (a_i \rightarrow m_i)$
$\bigwedge_j (m_j \rightarrow b_j)$
$[\forall m_i, \forall m_j, \forall n : \bigwedge_j (m_i < n) \bigwedge_j (n < m_j)$
$\rightarrow \exists m_k, \exists m_l : \bigwedge_k (m_k < n) \bigwedge_l (n < m_l) \bigwedge_i (c_k \rightarrow m_k)$
$\bigwedge_j (m_l \rightarrow d_l)$

The latter equation corresponds to the decomposition result produced by Alg. 1, i.e., the decomposition is lossless. Note that the decomposition is even stronger than the initial GCR, as it adds compliance on additional messages and enables distributed compliance checking.
In the following, we discuss the complexity of the GCR decomposition in Alg. 1. Results on checking regulatory compliance in general have been provided in [31]. The first and second loops iterate over the nodes of the compliance rule. If we consider that two nodes can only have one flow connector, then the number of required operations will be $n\frac{n-1}{2}$, otherwise $n(n-1)$. In both cases complexity is $O(n^2)$. The first if statement is $O(1)$, whereas the $else$ statement calculates $n\bullet$, $\bullet s$ and θ each with a worst case complexity of $O(n^2)$. The second inner loop has the same complexity as the first inner loop. The third nested inner loop iterates over partners and compare assertions within the same partner with a number of operations equal to $n \times m\frac{m-1}{2}$, which gives a complexity of $O(n^3)$. The last inner loop has a complexity of $O(n)$. Obviously, the overall worst case complexity of the algorithm is polynomial $O(n^4)$; i.e., outer loop combined with the third nested inner loop.

V. Implementation and Discussion

The presented approach was implemented as part of the C^3Pro framework[2], which deals with change and compliance in process choreographies [7]. The framework provides sophisticated functions for defining, propagating and negotiating changes in the context of process choreographies. Furthermore, it comprises a modeling component as one of its core components for editing and changing process changes, as well as for visualizing change propagations. In the context of the present work, the three-layer architecture of the framework [7] (i.e., process definition, change and execution) was extended with an additional layer dealing with compliance.
Figure 5 depicts the main components of the C^3Pro framework. The compliance and process modeling environments allow defining and editing *compliable* process choreography models [18], [20]. Compliability was introduced as *"a semantic correctness criterion to be considered when designing interaction models. It ensures that interaction models do not conflict with the set of imposed global compliance rules"* [20]. At design time, it is ensured that the created choreography models are compliant with the various compliance rules. The change editor allows defining and editing changes with respect to process models as well as compliance rules.

[2]http://www.wst.univie.ac.at/communities/c3pro/

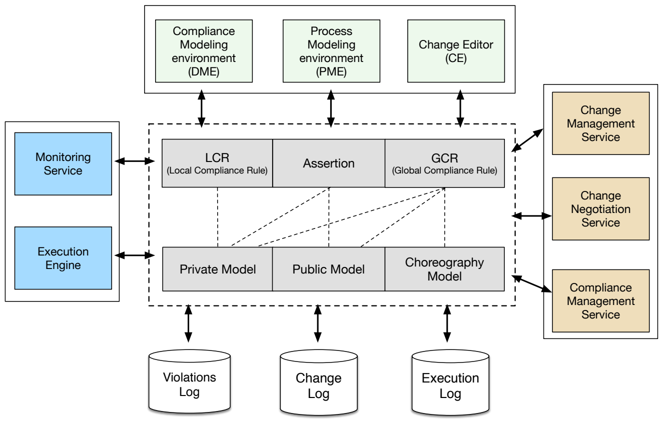
Compliance Modeling environment (DME)
Process Modeling environment (PME)
Change Editor (CE)
Monitoring Service
Execution Engine
LCR (Local Compliance Rule)
Assertion
GCR (Global Compliance Rule)
Private Model
Public Model
Choreography Model
Change Management Service
Change Negotiation Service
Compliance Management Service
Violations Log
Change Log
Execution Log

Fig. 5. C3Pro Prototype Architecture

The compliance management service handles the defined compliance rules and implements the GCR decomposition algorithm (cf. Section III-B). As process execution engine we utilized CPEE[3]. Most functions of the C^3Pro framework are provided as a RESTful service, which enables unified access from any client being able to communicate via HTTP. Finally, the Compliance Management Service serves as a pluggable middleware that may be used to integrate other process execution engines.
For testing the framework, we edited choreography and collaboration models in BPMN 2.0 using Signavio[4] and exported them to the C3Pro framework as XML files. Examples are extended with GCR, which are then decomposed into derived assertions using Alg. 1.
Applicability and discussion: The implementation of the decomposition algorithm takes two aspects into consideration: (i) the distributed execution of the processes, and (ii) the different visibility levels of tasks and control flow (privacy). The algorithm is initiated by the partners that share a GCR. As private tasks and their dependencies are not visible to all collaborators, parts of the decomposition algorithm are executed locally by all partners involved in this GCR to identify possible transitive relations between their corresponding private tasks and possible public tasks or interactions that replicate the connector where the GCR was split. This results in multiple derived assertion alternatives, which are then aggregated to alternatives from other partners in order to find a combination that recreates the original rule as described in Section III-B (cf. Example 1). Once the GCR is decomposed and the corresponding assertions are derived, each partner locally checks its derived assertions at runtime.
Note that this paper focuses solely on structural compliance. Compliance patterns that deal with data and resources are future work. In addition, the presence of ***XOR*** branches in the processes (where sending of messages on these branches is optional) does not affect the correctness of the decomposition as long as the processes are ***compliable*** with the original GCR

[3]http://cpee.org
[4]http://academic.signavio.com/

[20]. As aforementioned, we assume the soundness of the different process models (i.e., consistency and compatibility) and their compliability to the original GCR. This means that original GCR are correctly specified, and the decomposition enables their checking in a distributed way. In this case, transitivity ensures correct decomposition of GCR even at the presence of XOR branches. If no transitive relations are identified, then sync messages are required.

VI. Related Work

This work is positioned at the interface between process choreographies and process compliance. Process choreography research has been concerned with modeling choreographies and verifying their correctness (for an overview see [33]). Business process compliance, in turn, has been investigated for many years and several surveys exist (e.g., [3], [9]). In particular, existing approaches have dealt with compliance rule notations, including visual notations [2], [21], logic-based formalisms [24], and Event Calculus [27]. Moreover, approaches target at design and runtime compliance checking (see, e.g., [3], [23]) and different process perspectives such as time [30]. The following approaches address the interface between process choreographies and compliance: [15] advocates DCF Graphs for decomposing global contracts into local processes. [12] provides means to model contractual constraints. Compliance checking mechanisms assuming a trusted party are proposed by [11]. A distributed approach, which relies on IoT technology, is suggested by [26]. [34] assumes that partners try to provide wrong information and, hence, introduce the notion of accountability. [17] advocates compliance checking in process choreographies as crucial, but it cannot be assessed in how far the approach deals with the restricted visibility and availability of process information. In prior work, we have introduced the criterion of compliability [20] that captures the ability of a choreography to comply with a given set of compliance rules at all and how to check it [18]. [19] allows for checking effects of changes on compliance in process choreographies based on dependency graphs between global and local compliance rules as well as assertions. Finally, [8] provides an overview on the challenges, related approaches, and possible solutions at the interplay of compliance, change, and choreographies.

VII. Conclusion

The research question set out in the introduction is *how to decompose a GCR into derived assertions that can be checked in a distributed way without violating the privacy of the involved process partners?*

We provided a novel algorithm that decomposes GCR into assertions, which – as a particular benefit of the approach – can be checked by the partners in a distributed way and, hence, do not require any sensitive information to be exchanged at the choreography level. The correctness of the algorithm, and hence a formal evaluation, was accomplished as follows: based on the derived assertions the original GCR (or a stricter one) can be reconstructed. As a current limitation we assume that GCRs consist of exactly one antecedence pattern, i.e., a pattern triggering the rule. While a collection of scenarios is supported by such rules, future work will extend the algorithm and the formal proof to GCR with an arbitrary number of antecedence patterns. In future work, we will further elaborate a full-blown prototype and demonstration environment including the implementation of real-world scenarios. As a promising application area we have identified the manufacturing domain as indicated by the running example. Others include health-care and logistics where complex networks are subject to a multitude of regulations and demand for adaptations at the same time. Furthermore, we plan to adapt our algorithms to other languages that are used to specify compliance rules (e.g., Declare [28], PENELOPE [10], or BPMN-Q [2]).

References

[1] van der Aalst, W.M.P., Lohmann, N., Massuthe, P., Stahl, C., Wolf, K.: Multiparty contracts: Agreeing and implementing interorganizational processes. Comput. J. 53(1), 90–106 (2010)

[2] Awad, A., Weidlich, M., Weske, M.: Visually specifying compliance rules and explaining their violations for business processes. Journal of Visual Languages & Computing 22(1), 30–55 (2011)

[3] Becker, J., Delfmann, P., Eggert, M., Schwittay, S.: Generalizability and applicability of model-based business process compliance-checking approaches - a state-of-the-art analysis and research roadmap. Bus Res 5(2), 221–247 (2012)

[4] Bischoff, F., Fdhila, W., Rinderle-Ma, S.: Generation and transformation of compliant process collaboration models to BPMN. In: Advanced Information Systems Engineering. pp. 462–478 (2019)

[5] Ciccio, C.D., Maggi, F.M., Montali, M., Mendling, J.: Resolving inconsistencies and redundancies in declarative process models. Inf. Syst. 64, 425–446 (2017)

[6] Fdhila, W., Gall, M., Rinderle-Ma, S., Mangler, J., Indiono, C.: Classification and formalization of instance-spanning constraints in process-driven applications. In: BPM. pp. 348–364 (2016)

[7] Fdhila, W., Indiono, C., Rinderle-Ma, S., Reichert, M.: Dealing with change in process choreographies: Design and implementation of propagation algorithms. Inf Sys 49, 1 – 24 (2015)

[8] Fdhila, W., Rinderle-Ma, S., Knuplesch, D., Reichert, M.: Change and compliance in collaborative processes. In: SCC. pp. 162–169 (2015)

[9] Fellmann, M., Zasada, A.: State-of-the-art of business process compliance approaches. In: ECIS (2014)

[10] Goedertier, S., Vanthienen, J.: Designing compliant business processes with obligations and permissions. In: BPM'06 Workshops. pp. 5–14 (2006)

[11] González, L., Ruggia, R.: A comprehensive approach to compliance management in inter-organizational service integration platforms. In: ICSOFT. pp. 722–730 (2018)

[12] Governatori, G., Idelberger, F., Milosevic, Z., Riveret, R., Sartor, G., Xu, X.: On legal contracts, imperative and declarative smart contracts, and blockchain systems. Artif. Intell. Law 26(4), 377–409 (2018)

[13] Hashmi, M., Governatori, G., Lam, H., Wynn, M.T.: Are we done with business process compliance: state of the art and challenges ahead. Knowl. Inf. Syst. 57(1), 79–133 (2018)

[14] Hashmi, M., Governatori, G., Wynn, M.T.: Normative requirements for regulatory compliance: An abstract formal framework. Information Systems Frontiers 18(3), 429–455 (2016)

[15] Hildebrandt, T.T., Mukkamala, R.R., Slaats, T., Zanitti, F.: Contracts for cross-organizational workflows as timed dynamic condition response graphs. J. Log. Algebraic Methods Program. 82(5-7), 164–185 (2013)

[16] Kasse, J.P., Xu, L., de Vrieze, P., Bai, Y.: The need for compliance verification in collaborative business processes. In: PRO-VE. pp. 217–229 (2018)

[17] Kasse, J.P., Xu, L., de Vrieze, P., Bai, Y.: Verifying for compliance to data constraints in collaborative business processes. In: PRO-VE. pp. 259–270 (2019)

[18] Knuplesch, D., Reichert, M., Pryss, R., Fdhila, W., Rinderle-Ma, S.: Ensuring compliance of distributed and collaborative workflows. In: CollaborateCom'13. pp. 133–142 (2013)

[19] Knuplesch, D., Fdhila, W., Reichert, M., Rinderle-Ma, S.: Detecting the effects of changes on the compliance of cross-organizational business processes. In: ER'15. pp. 94–107 (2015)

[20] Knuplesch, D., Reichert, M., Fdhila, W., Rinderle-Ma, S.: On enabling compliance of cross-organizational business processes. In: BPM, pp. 146–154 (2013)

[21] Knuplesch, D., Reichert, M., Kumar, A.: A framework for visually monitoring business process compliance. Inf. Syst. 64, 381–409 (2017)

[22] Knuplesch, D., Reichert, M., Ly, L.T., Kumar, A., Rinderle-Ma, S.: On the formal semantics of the extended compliance rule graph. Tech. Rep. 2013-05, Ulm University (2013)

[23] Ly, L.T., Maggi, F.M., Montali, M., Rinderle-Ma, S., van der Aalst, W.M.P.: Compliance monitoring in business processes: Functionalities, application, and tool-support. Inf. Syst. 54, 209–234 (2015)

[24] Maggi, F., Montali, M., Westergaard, M., van der Aalst, W.M.P.: Monitoring business constraints with linear temporal logic: an approach based on colored automata. In: BPM. pp. 132–147 (2011)

[25] Mendling, J., et al.: Blockchains for business process management - challenges and opportunities. ACM Trans. Management Inf. Syst. 9(1), 4:1–4:16 (2018)

[26] Meroni, G., Baresi, L., Montali, M., Plebani, P.: Multi-party business process compliance monitoring through iot-enabled artifacts. Inf. Syst. 73, 61–78 (2018)

[27] Montali, M., Maggi, F.M., Chesani, F., Mello, P., van der Aalst, W.M.P.: Monitoring business constraints with the event calculus. Trans on Intelligent Sys and Tech 5(1), 17.1–17.30 (2014)

[28] Pesic, M., Schonenberg, H., van der Aalst, W.M.P.: DECLARE: full support for loosely-structured processes. In: EDOC. pp. 287–300 (2007)

[29] Schunselaar, D.M.M., Maggi, F.M., Sidorova, N.: Patterns for a log-based strengthening of declarative compliance models. In: Integrated Formal Methods. pp. 327–342 (2012)

[30] Taghiabadi, E.R., Fahland, D., van Dongen, B.F., van der Aalst, W.M.P.: Diagnostic information for compliance checking of temporal compliance requirements. In: Advanced Information Systems Engineering. pp. 304–320 (2013)

[31] Tosatto, S.C., Governatori, G., van Beest, N.: Checking regulatory compliance: Will we live to see it? In: Business Process Management. pp. 119–138 (2019)

[32] Tosatto, S.C., Governatori, G., van Beest, N., Olivieri, F.: Efficient full compliance checking of concurrent components for business process models. FLAP 6(5), 963–998 (2019)

[33] Weske, M.: Business Process Management - Concepts, Languages, Architectures, Third Edition. Springer (2019)

[34] Yao, J., Chen, S., Levy, D.: Accountability-based compliance control of collaborative business processes in cloud systems. In: Security, Privacy and Trust in Cloud Systems, pp. 345–374 (2014)

APPENDIX

A. Choreography

Definition 3 (Choreography, slightly adapted from [7]): We define a choreography y as a tuple ($\mathcal{P}$, $\mathcal{G}$ Π, $\mathcal{L}$, ψ, Γ, ξ) where,

- $\mathcal{P}$ is the set of all participating partners.
- $\mathcal{G}$ is the choreography model representing the interactions $\mathcal{I}$ between partners in $\mathcal{P}$ (cf. Fig. 1).
- $\Pi = \{\pi_p\}_{p \in \mathcal{P}}$ is the set of all private models.
- $\mathcal{L} = \{l_p\}_{p \in \mathcal{P}}$ is the set of all public models.
- $\psi = \{\psi_p : l_p \leftrightarrow \pi_p\}_{p \in \mathcal{P}}$ is a partial mapping function between nodes of the public and private models.
- Γ: $l \leftrightarrow l'$ is a partial mapping function between nodes of different public models.
- $\xi : \mathcal{G} \leftrightarrow l$ is a partial mapping function between nodes of the choreography model and the public models.

Based on functions ψ and Γ, certain soundness properties of choreography y can be checked, including *structural and behavioral compatibility* between public models and *consistency* between public and private models [7].

B. Additional Transitivity Proofs

Proof 3 (Chaining Transitivity): Let A, B, C be three activities or interactions such as $\boxed{A} \mapsto \boxed{B} \wedge \boxed{A} \dashrightarrow \textcircled{C} \wedge \textcircled{C} \dashrightarrow \boxed{B}$: if A and B occur then C should occur in between.

Let Let M, E, F be three activities or interactions such as

- (1) $\boxed{A} \mapsto \boxed{M} \wedge \boxed{A} \dashrightarrow \textcircled{E}$
- (2) $\boxed{M} \mapsto \boxed{B} \wedge \boxed{F} \dashrightarrow \textcircled{B}$
- (3) $\boxed{E} \mapsto \boxed{F} \wedge \boxed{E} \dashrightarrow \textcircled{C} \wedge \textcircled{C} \dashrightarrow \boxed{F}$

Then, $(1) \wedge (2) \wedge (3)$

$:\Leftrightarrow \quad \forall a \forall m : \Big((x(a, A) \wedge (x(m, M) \wedge a < m) \rightarrow \exists e : (x(e, E) \wedge a < e)\Big) \wedge$

$\forall m \forall b : \Big((x(m, M) \wedge (x(b, B) \wedge m < b) \rightarrow \exists f : (x(f, F) \wedge f < b)\Big) \wedge$

$\forall e \forall f : \Big((x(e, E) \wedge (x(f, F) \wedge e < f) \rightarrow \exists c : (x(c, C) \wedge e < c \wedge c < f)\Big)$

$\Leftrightarrow \ \forall a \forall m \forall b : \Big((x(a, A) \wedge x(m, M) \wedge x(b, B) \wedge a < m \wedge m < b) \rightarrow (\exists e \exists f((x(e, E) \wedge x(f, F) \wedge a < e \wedge f < b)\Big) \wedge$

$\forall e \forall f : \Big((x(e, E) \wedge (x(f, F) \wedge e < f) \rightarrow \exists c : (x(c, C) \wedge e < c \wedge c < f)\Big)$

$(using\ Theorem\ 1) \Leftrightarrow \ \forall a \forall b : \Big((x(a, A) \wedge x(b, B) \wedge a < b) \rightarrow \exists e \exists f : (x(e, E) \wedge x(f, F) \wedge a < e \wedge f < b)\Big) \wedge$

$\forall e \forall f : \Big((x(e, E) \wedge (x(f, F) \wedge e < f) \rightarrow \exists c : (x(c, C) \wedge e < c \wedge c < f)\Big)$

$\Rightarrow \quad \forall a \forall b : \Big((x(a, A) \wedge x(b, B) \wedge a < b) \rightarrow \exists e \exists f \exists c : (x(e, E) \wedge x(f, F) \wedge x(c, C) \wedge a < e \wedge e < c \wedge c < f \wedge f < b)\Big)$

$\Rightarrow \ \forall a \forall b : \Big((x(a, A) \wedge x(b, B) \wedge a < b) \rightarrow \exists c : (x(c, C) \wedge a < c \wedge c < b)\Big)$

Proof 4 (Requires transitivity): Let A, B be two activities or interactions such as $\boxed{A} \dashrightarrow \textcircled{B} \vee \textcircled{B} \dashrightarrow \boxed{A}$: if A occurs then B should occur (before or after, $\exists A \rightarrow \exists B$):.

Let A, B, M be three activities or interactions such as $(\boxed{A} \dashrightarrow \textcircled{M} \vee \textcircled{M} \dashrightarrow \boxed{A}) \wedge (\boxed{M} \dashrightarrow \textcircled{B} \vee \textcircled{B} \dashrightarrow \boxed{M})$. Then

$:\Leftrightarrow \ \exists a : \Big(x(a, A) \rightarrow \exists m : x(m, M))\Big) \wedge \exists m : \Big(x(m, M) \rightarrow \exists b : x(b, B))\Big)$

$\Leftrightarrow \ \exists a \Big(x(a, A) \rightarrow \exists b : x(b, B)\Big)$