Semantic Conflict Prevention in Service Choreography

Young Yoon, Chunyang Ye, and Hans-Arno Jacobsen

Middleware Systems Research Group, University of Toronto

Abstract. The choreography of business processes deployed across different organizations on large-scale enterprise networks is prone to producing semantically conflicting behavior. For example, increasing production is inconsistent with a sales promotion given the business goal of clearing out the inventory. This kind of conflicting behavior among collaborating partners is usually caused by a non-realizable global specification that is unable to control the partners' interaction in a complete and consistent way. To address this issue, we propose a framework to discover and fix a problematic global specification by introducing extra coordination mechanisms. We prove that the resulting global specification can be realized by a set of local specifications that guarantee consistent behavior among partners. Experimental results show that our approach is promising in preventing semantic conflicts in the choreography of business processes with a reasonable overhead.

1 Introduction

The service-oriented architecture (SOA) is an emerging software engineering paradigm for developing distributed collaborative enterprise applications. In SOA, business processes are encapsulated as Web services and published to service consumers. By selecting suitable services from different organizations, invoking them and choreographing their interactions, service consumers can easily and quickly develop service-oriented applications. However, the choreography of distributed business processes that are deployed across different organizations on large-scale enterprise networks is prone to producing semantically conflicting behavior. For example, increasing production is inconsistent with a sales promotion given the business goal of clearing out the inventory[1].This kind of phenomenon is usually referred to as a *semantic conflict* among business processes [16].

Semantic conflicts are unacceptable for service-oriented applications and detrimental to business operations. It is therefore desirable for designers to specify a *global specification* that controls the interactions among business processes to prevent conflicting scenarios at runtime. For example, the designer may write a global specification for the aforementioned example to safeguard the application from entering into the conflicting scenario by specifying some constraints that the interactions among the business processes should obey. Since services are usually deployed and executed in a distributed and autonomous manner, a global specification needs to be enforced collaboratively by partner services. To do so, each organization needs to execute the part of the global specification that relates to its interaction in the choreography. We call the part of the global

[1] The example was introduced to us as a real-world use case in a guest lecture in 2009 [23].

specification specific to a particular organization a *local specification*. To guarantee that all local specifications conform to the global specification, much existing research has been devoted to verify the consistency between all local and the global specification [2, 11, 12, 18].

An assumption underlying these approaches is that the global specification is realizable *i.e.,* there exists a set of local specifications that can implement and conform to the behavior of the global specification. But, such an assumption may not always hold in practice. Note that the non-realizability of the global specification is not uncommon in software engineering for many reasons [4, 13]. In particular, the global specification for a business choreography may be strictly based on very limited or incomplete requirements that do not properly account for hidden implications in distributed collaborations. Also, a service developer may focus only on updating part of global specification and overlook whether the change is consistent with the overall specification. Furthermore, changes can be ad-hoc due to the highly dynamic nature of the distributed collaboration, thus it is hard to control the quality. These faulty operations and maintenance contributed by designers or the collaborating partners may yield a problematic non-realizable global specification. As a result, even though a global specification is implemented and enforced by a set of local specifications, the conflicting scenarios may still occur at runtime.

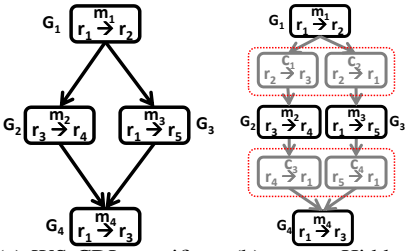
One solution to this issue is to check whether a global specification is realizable before implementing and deploying it to distributed organizations. However, checking a global specification is realizable or not is undecidable in general [4, 13]. Hence, this approach can be applied to only a restricted subset of applications. Moreover, such a solution does not provide guidance and hints to fix the problematic global specification.

In this paper, we propose a different approach to address this issue. Instead of checking whether a global specification is realizable or not, our approach analyzes and discovers the potential conflicting scenarios in a global specification and enriches the potentially problematic global specification with extra coordination information. The purpose is to remind service consumers of the potential risks in a global specification and recommend solutions to fix loopholes in the global specification. Our major contribution is a framework to extract and derive hidden synchronizability constraints in a global specification, automatically (Section 3 - 4) The fully implemented framework is formally proven to be free of semantic conflicts with a theoretical model based on the Pi-Calculus (Section 2). Empirical evidence based on extensive experimentation exhibits no significant additional overhead when the framework is used (Section 5).

2 Service Choreography

In this section, we briefly introduce the concept of service choreography, and its conventional implementation with examples. Then, we specify the concepts and discussions in a formal model based on the Pi-calculus [17], which is the theoretical basis of service choreography. The service choreography for a service composition, also known as the global specification for a service composition, is usually specified in WS-CDL [22], which describes the interactions among collaborating business processes, as shown abstractly in Figure 1(a).

In this model, the collaboration among partners are organized as a process, and each task in the process is called an *interaction*, which specifies a communication between a pair of partners r_i and r_j, such that r_i sends a message m to r_j. We formalize the concepts of service choreography using pi-calculus[2] as follows:



(a) WS-CDL specification (b) Hidden synchronizability

Fig. 1. Global specification of service choreography.

Definition 1. *A global specification GS is defined as follows:*

$$GS \equiv (a_{int.} \cdot GS) \mid (GS \parallel GS) \mid (GS + GS) \mid 0$$

where $a_{int.} \equiv r_1 \overset{m}{\Rightarrow} r_2$ represents an interaction i.e.*, r_1 sends a message m to r_2. Operators '$\cdot$', '$\parallel$', and '+' represent the prefix, parallel, and choice operators, respectively. The termination process is represented as* 0.

In practice, a service consumer usually designs the global specification to govern the collaboration at an abstract level. Since the involved business processes are usually distributed across different organizations, the global specification needs to be implemented by the partners in a distributed way. Conventionally, the global specification for a service choreography is decomposed into a set of local specifications. These local specifications are implemented and deployed to each partner, which is formalized as follows:

Definition 2. *A local specification LS is defined as follows:*

$$LS \equiv (x(m) \cdot LS) \mid (\bar{x}(m) \cdot LS) \mid (LS \parallel LS) \mid (LS + LS) \mid 0$$

where $x(m)$ and $\bar{x}(m)$ represent sending and receiving a message m, respectively.

Given a global specification, the decomposition of the global specification is to implement all the specified interactions in the local specifications. Before introducing the conventional decomposition algorithm, let us first define some concepts. Given a process P, if P executes an action a and transitions to another state P', then we denote it as $P \xrightarrow{a} P'$. Based on this notation, we can describe the semantics of operators '$\cdot$', '$\parallel$', and '+' as follows:

$$a \cdot P \xrightarrow{a} P \qquad \frac{P \xrightarrow{a} P'}{P+Q \xrightarrow{a} P'+Q} \qquad \frac{Q \xrightarrow{a} Q'}{P+Q \xrightarrow{a} P+Q'}$$

$$\frac{P \xrightarrow{x(m)} P', Q \xrightarrow{\bar{x}(m)} Q'}{P \parallel Q \xrightarrow{a_{int.}} P' \parallel Q'} \qquad \frac{P \xrightarrow{a_{int.}} P'}{P \parallel Q \xrightarrow{a_{int.}} P' \parallel Q} \qquad \frac{Q \xrightarrow{a_{int.}} Q'}{P \parallel Q \xrightarrow{a_{int.}} P \parallel Q'}$$

With the semantics of these operators, we can define the trace of a process as follows:

[2] To ease the presentation and simplify the model, only a subset of the pi-calculus syntax is adopted

Definition 3. *Given a process P, a trace of P is defined as a sequence of actions a_1, a_2, $\cdots$, a_n, satisfying the following condition: $\exists P_1, P_2, \cdots, P_n : P \xrightarrow{a_1} P_1$, $P_1 \xrightarrow{a_2} P_2, \cdots$, $P_{n-1} \xrightarrow{a_n} P_n$. To ease the presentation, we denote $trace(P)$ as the set of all traces of P.*

Using above notations, the decomposition of a global specification into corresponding local specifications is briefly described below:

Definition 4. *Let GS be a global specification for a service choreography and r_i be an involved partner. The local specification of r_i derived from GS, denoted as $LS_{ri} = \varphi(GS, r_i)$, can be derived recursively by the following rules :*

1. $\varphi(a_{int.} \cdot GS, r_i) = x(m) \cdot \varphi(GS, r_i), if\ a_{int.} \equiv r_i \overset{m}{\Rightarrow} r_j$
2. $\varphi(a_{int.} \cdot GS, r_j) = \bar{x}(m) \cdot \varphi(GS, r_j), if\ a_{int.} \equiv r_i \overset{m}{\Rightarrow} r_j$
3. $\varphi(a_{int.} \cdot GS, r_i) = \varphi(GS, r_i), if\ a_{int.} \equiv r_j \overset{m}{\Rightarrow} r_k (j, k \neq i)$
4. $\varphi(GS_1 + GS_2, r_i) = \varphi(GS_1, r_i) + \varphi(GS_2, r_i)$
5. $\varphi(GS_1 \parallel GS_2, r_i) = \varphi(GS_1, r_i) \parallel \varphi(GS_2, r_i)$
6. $\varphi(0, r_i) = 0$

These rules illustrate how to project a global specification into local specifications. In particular, Rule 1 and Rule 2 say that if r_i interacts with r_j, then in their local specifications, r_i needs to send a message m and r_j needs to receive this message. Rule 3 means that for all the other partners than r_i and r_j, they need not send or receive any message related to this interaction. Rule 4 and Rule 5 represent that the decomposition algorithm of a global specification can be constructed using divide-and-conquer principle. Rule 6 says that no local specification is needed for an empty global specification (that is, no constraints for the partners involved in the service composition).

The construction of the local specifications is straightforward and syntactically correct. However, the service choreography in production may still exhibit behavior that does not conform exactly to the original global specification. To guarantee that these derived local specifications correctly implement the global specification, the global specification should be realizable.

Definition 5. *Given a global specification GS, if there exists a set of local specifications LS_1, LS_2, $\cdots$, LS_m, such that $trace(LS_1 \parallel LS_2 \parallel \cdots \parallel LS_m) = trace(GS)$, then GS is called realizable.*

Note that not all global specifications are realizable as shown in the following counter example.

Suppose $GS = b_{int.} \cdot 0 + c_{int.} \cdot 0$, where $b_{int.} \equiv r_1 \overset{m_1}{\Rightarrow} r_2$, $c_{int.} \equiv r_2 \overset{m_2}{\Rightarrow} r_1$, The corresponding local specifications are: $LS_{r1} \equiv x_1(m_1) \cdot 0 + \bar{x}_2(m_2) \cdot 0$, $LS_{r2} \equiv \bar{x}_1(m_1) \cdot 0 + x_2(m_2) \cdot 0$. Hence, $LS_{r1} \parallel LS_{r2} \equiv b_{int.} \cdot 0 + c_{int.} \cdot 0 + x_1(m_1) \cdot x_2(m_2) \cdot 0 + x_2(m_2) \cdot x_1(m_1) \cdot 0$. Therefore, $trace(GS) \neq trace(LS_{r1} \parallel LS_{r2})$.

The rest of the paper discovers the reason for non-realizability, and presents solutions to prevent it.

3 Problem Statement

In a service choreography, services interact with each other autonomously, typically in a distributed manner. As mentioned earlier, in such distributed nature of the service

choreography, a service may exhibit a behavior semantically conflicting with other services, unless there is a mechanism to govern every service's behavior to meet the constraints specified in the service choreography. On the other hand, services may evolve from time to time to react to changing requirements, which can affect the realizability of the global specification. The challenge of implementing such a control mechanism runs deeper than the problem of satisfying the requirement of operating the mechanism scalably. The difficulty stems from the reality that a global specification of a service choreography can be incomplete in the first place, *i.e.*, the scoping of the allowable behavior for the interactions among the involved services is not explicitly specified or it is simply overlooked. Thus, given the problematic global specification, the control mechanism is not guaranteed to be robust.

The first type of semantic conflict the control mechanism can overlook is the violation of message ordering. For example, considering the root of the global specification to be a pick[3] in Figure 1(a), the following message sequences are expected from the choreography: m_1, m_2, m_4 or m_1, m_3, m_4. However, the decomposed local specifications may produce different message patterns. For instance, even though r_1 is the initial sender among all that starts the global process as in Figure 2(a), nothing prevents r_3 from starting its local process arbitrarily with the task of sending m_2 before r_1 sends out m_1. The ordering of the messages is clearly violated, making the global specification non-realizable.

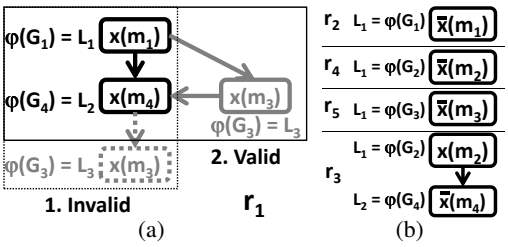


Fig. 2. Incorrect decomposition of global specification.

Another common type of semantic conflict is pick violation. Revisit the example in Figure 1(a). Once G_2 is picked the message sequence, m_2, m_4, is expected. However, r_1 in G_3 can still be triggered to send m_3, unless there is synchronous coordination of picks. In fact, the constraints to avoid such conflicts are hidden as shown in Figure 1(b), *e.g.*, m_1 should strictly precede m_2 or m_3 by a confirmation message to be sent from r_2 of G_1 to r_3 of G_2. The global specification cannot be realizable unless the hidden constraints are discovered and extracted. So, one may resort to the solution that recommends a set of best practices for revising a problematic global specification manually. However, it is not only burdensome but also error-prone for designers to do so. Instead, we provide an *automated* framework that implements the functionality stated in the following formal definition of our problem.

Problem Definition. Given a global specification GS, define a function $GS' = \Omega(GS)$ such that GS' is realizable.

Our novel framework discovers potential semantic conflicts from an incomplete or incorrect global specification of a service chorography. The conflicts are avoided by a

[3] Pick and flow are branching operations specified in WS-CDL[22] According to our formal model, pick is '+' and flow is '$\|$'.

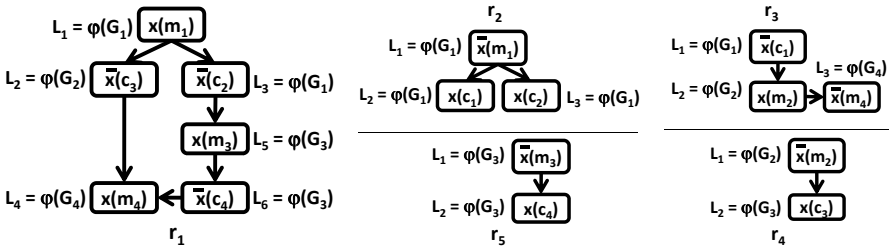


Fig. 3. Example of reliable decomposition

synchronization scheme which is automatically derived and incorporated into the local specifications for the involved partners through a refined decomposition procedure. Our objective is to guarantee that the constraints for the implicit behaviors are always satisfied transparently without the need to incur any significant refactoring of the existing service choreography.

4 Reliable Service Choreography

In this section, we provide both abstract and practical solutions for a reliable service choreography.

4.1 Overview

So far, we learned that the non-realizable global specifications cannot be implemented into a reliable service choreography. The reason is that synchronizability relationship between the involved partners are missing in the non-realizable global specifications. The lack of synchronizability relationship in the global specifications may be caused by incomplete description of the service interaction, or unawareness of the implicit synchronization requirement among the distributed partners, or unexpected impacts from the changes during the choreography evolution. For instance, an originally realizable specification, $GS = b_{int.} \cdot 0$, becomes non-realizable after it is evolved to $GS = b_{int.} \cdot 0 + c_{int.} \cdot 0$. Therefore, we present a solution that automatically derives the missing synchronizability and explicitly incorporate it among the partners.

To sketch out the new solution, let's revisit the sample global specification in Figure 1. The global specification is decomposed into the set of decomposed local specifications for r_1 - r_5 in the same order as in the previous conventional decomposition, *i.e.*, $G_1 \rightarrow G_2 \rightarrow G_4 \rightarrow G_3$. Notice the key difference between Figure 3 and Figure 2 is that coordination messages, c_i, where $i = 1, 2, 3, 4$, are derived and mapped to appropriate partners. For example, between G_1 and G_2, a coordination is necessary as the global specification implicitly states the constraint that r_3 can send out message m_2, *iff* r_2 completes the receipt of the message m_1 from r_2. In this particular example, every run of the service choreography through the decomposed set of local specifications with the introduction of explicit coordination among partners guarantees to yield the valid message sequences that are implicitly specified by Figure 1(b): m_1, c_1, m_2, c_3, m_4 or

m_1, c_2, m_3, c_4, m_4, mainly because there is a unique global initial task in the decomposed choreography which is $x(m_1)$ by r_1.

We formally explain the mechanism behind the guaranteed conformance to the global specification in Section 4.2 and we present the practical implementation of the mechanism in Section 4.3.

4.2 Conflict Detection and Resolution

This section presents the abstract rules for discovering the missing synchronizability in the global specification and reflecting it to the local specification.

Definition 6. *Let GS be a global specification and $LS_{ri} \equiv \varphi(GS, r_i)$, a conflict is defined as a trace $a_1,a_2,\cdots,a_k$ such that $a_1,a_2,\cdots,a_k \in trace(LS_{r1} \parallel \cdots \parallel LS_{rm}) \wedge a_1,a_2,\cdots,a_k \notin trace(GS) \wedge a_1,a_2,\cdots,a_{k-1} \in trace(GS)$.*

Note that such abnormal scenarios are useful for service developers for detecting the conflicts and for diagnosing the sources of conflicts. Due to limited space, we provide a framework in our technical report [23] to analyze such conflicting scenarios that may happen for a global specification. On the other hand, to *prevent* such conflicting scenarios, we need to incorporate the missing synchronizations among the partners in the global specification. The following rules show how to do that:

Definition 7. *Let GS be a global specification, and $GS' \equiv \Omega(GS)$ is defined recursively by the following rules:*

1. $\Omega(a_{int.} \cdot b_{int.} \cdot GS) = a_{int.} \cdot c_{int.} \cdot \Omega(b_{int.} \cdot GS)$, *where* $a_{int.} \equiv r_1 \overset{m_1}{\Rightarrow} r_2, b_{int.} \equiv r_3 \overset{m_2}{\Rightarrow} r_4, r_2 \neq r_3$, $c_{int.} \equiv r_2 \overset{c_1}{\Rightarrow} r_3$.
2. $\Omega(a_{int.} \cdot b_{int.} \cdot GS) = a_{int.} \cdot \Omega(b_{int.} \cdot GS)$, *where* $a_{int.} \equiv r_1 \overset{m_1}{\Rightarrow} r_2, b_{int.} \equiv r_3 \overset{m_2}{\Rightarrow} r_4, r_2 = r_3$.
3. $\Omega(a_{int.} \cdot GS_1 + b_{int.} \cdot GS_2) = c_{int.1} \cdot \Omega(a_{int.} \cdot GS_1) + c_{int.2} \cdot \Omega(b_{int.} \cdot GS_2)$, *where* $a_{int.} \equiv r_1 \overset{m_1}{\Rightarrow} r_2, b_{int.} \equiv r_3 \overset{m_2}{\Rightarrow} r_4, r_1 \neq r_3$, $c_{int.1} \equiv r_c \overset{c_1}{\Rightarrow} r_1$, $c_{int.2} \equiv r_c \overset{c_2}{\Rightarrow} r_3$.
4. $\Omega(a_{int.} \cdot GS_1 + b_{int.} \cdot GS_2) = \Omega(a_{int.} \cdot GS_1) + \Omega(b_{int.} \cdot GS_2)$, *where* $a_{int.} \equiv r_1 \overset{m_1}{\Rightarrow} r_2, b_{int.} \equiv r_3 \overset{m_2}{\Rightarrow} r_4, r_1 = r_3$.
5. $\Omega(GS_1 \parallel GS_2) = \Omega(GS_1) \parallel \Omega(GS_2)$.
6. $\Omega(a_{int.} \cdot 0) = a_{int.} \cdot 0$
7. $\Omega(0) = 0$.

Based on these rules in Definition 7, we can enrich a global specification into a realizable one with extra coordination messages and the coordinator. In particular, Rule 1 addresses the out-of-order messages by adding a synchronization message between two consecutive global tasks where the receiver in the first one is different from the sender in the second one. Rule 3 addresses the invalid pick problem by adding a coordinator to coordinate the two branches if the senders in the different branches belong to different partners. Rule 2 states that if the receiver in an interaction is also the sender in the immediately following interaction, then no synchronization is needed. Similarly, if the senders in the first interactions in two branches are from the same partner, then no synchronization is needed. In addition, Rule 5 says that the two global specifications can be enriched individually in parallel. Rules 6 and 7 specify the termination conditions of enriching a global specification. The following theorem guarantees the correctness of our approach.

Algorithm 1: Decomposition(G)

Input: Global specification $\mathbb{GS}$,
Set of partners $\mathbb{R} = \{r_i | i = 1, 2, ...n\}$
Output: Set of local specifications $mathbbLS$

1 **if** *G is not decomposed* **then**
2 **if** *G is a successor of junction node* **then**
3 **foreach** $r \in \mathbb{R}$ **do**
4 $CurrentNode\_r$ = junction node;
5 $(LS\_currentSender)\_G$.AddTask(G.id(), $x(current\_message)$);
6 $(LS\_currentReceiver)\_G$.AddTask(G.id(), $\overline{x}(current\_message)$);
7 **foreach** *nextTask of G* **do**
8 **if** *(nextTask.sender $\neq$ currentReceiver) or (G is $PICK$)* **then**
9 $LS\_currentReceiver$.AddTask(G.id(), $x(c)$);
10 $LS\_(nextTask\_sender)$.AddTask(G.id(), $\overline{x}(c)$);
11 **foreach** *nextTask of G* **do**
12 **if** *nextTask is not decomposed* **then**
13 Decomposition(nextTask);
14 **else**
15 $LS\_(nextTask\_sender)$.AddTransition(nextTask.id());
16 **if** *currentSender = nextReceiver | currentReceiver = nextReceiver* **then**
17 $LS\_(nextTask\_receiver)$.AddTransition(nextTask.id());

Algorithm 2: AddTask(id, L_{new})

Input: Local process $\mathbb{LS}$,
$Hash(k, List)$ where k = global task id,
$Last\_k = 0$
1 Hash.put(id, L_{new});
2 **if** *L_{new} is PICK or FLOW* **then**
3 $L_{last} = Hash.get(Last\_k).getFirstElement()$;
4 **else**
5 $L_{last} = Hash.get(Last\_k).getLastElement()$;
6 $Succ\_L_{last} \leftarrow Succ\_L_{last} \cup L_{new}.id()$;
7 $Prec\_L_{new} \leftarrow Prec\_L_{new} \cup L_{last}.id()$;
8 $Last\_k \leftarrow LT_{new}.id()$;

Algorithm 3: AddTransition(id)

Input: $Hash(k, List(L, L'))$ where k = global task id,
$Last\_k$
1 $L_{from} = Hash.get(id).getLastElement()$;
2 $L_{to} = Hash.get(Last\_k).getLastElement()$;
3 $Prec\_(L_{from}) \leftarrow Prec\_(L_{from}) \cup L_{to}.id()$;
4 $Succ\_(L_{from}) \leftarrow Succ\_(L_{from}) \cup L_{from}.id()$;

Theorem 1. *Given a global specification GS and $GS' \equiv \Omega(GS)$, GS' is realizable, that is, $trace(GS')$ = $trace(LS_{r1} \parallel \cdots \parallel LS_{rm})$, where $LS_{ri} \equiv \varphi(GS', r_i)$.*

The inductive proof of Theorem 1 is based on the rules in Definition 7. The full proof is provided in our technical report [23]. In the next section, we design algorithms to implement these rules.

4.3 Reliable Decomposition Framework

In the previous section, we explained at a high level that the new coordination mechanism guarantees to prevent the conflicts. In this section, we present the practical techniques to implement the coordination mechanism. Algorithm 1 presents the arrangement of coordination messages between partners. Suppose the two subsequent global tasks, G followed by G', are given. If the receiver of G is the different partner as the sender of G', then the algorithm must let the receiver of G to include a local task of sending out a coordination message toward the sender of G' into its local specification as in (Algorithm 1: 6 - 10). Unlike the conventional faulty decomposition methods,

the multiple local tasks can be produced by a single projection of a global task (Algorithm 1: 6 - 7) for a sender and a receiver. The new mapping is maintained in a hash table (Algorithm 2: Input).

First, the local task of sending a coordination message should make a transition from the branching task (pick/flow) in the local specification(Algorithm 3:2-3), if the local task is projected by a global branching task, as r_2 in Figure 3. When the decomposition framework explicitly requests (Algorithm 1:15-17) to set up a transition using the $AddTransition$ (Algorithm 3) from the last added node to a node projected by the specified global task, then there is a choice to be made to which element should the last added node make a transition to, because the mapping returns a list of local tasks. Our framework simply chooses the last element (Algorithm 3:2) and adds the transition by updating the pointers to immediately preceding or succeeding nodes (Algorithm 3:3-4). The newly derived service choreography, for example as in Figure 3, always produce the valid message sequences thanks to the strict serialization through explicit arrangement of coordination among partners. Particularly for Figure 3, there is only one unique initial send task (L_1 of r_1) globally and all others are under the constraint to start to send a message only after receiving some message first.

Special attention must be paid for *pick* as illustrated in Figure 4. Note that Figure 4 is different from Figure 1(a) that the receiver of G_1 and the sender of G_2 are actually the same (r_2). Hence, upon the execution of G_2, r_2 does not need to wait for a coordination message. However, r_2 can proceed to the next local task of sending the message m_2 even in the case G_3 was picked instead right after the execution of G_1, which is a clear violation of pick and it causes a conflict. This calls for a dedicated coordinator to handle picks at the branching task. At the branching task, the framework must include local tasks of sending coordination messages to the coordinator into the local process of receiver. The coordinator relays the coordination message to the sender of the succeeding global task. So pick is reliably executed, since the sender of the succeeding global task always wait for the coordination message from the coordinator no matter what receiver of the preceding task is.

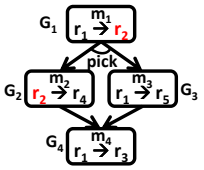


Fig. 4. Pick violation.

5 Evaluation

This section presents experimental evaluation of the reliable global process decomposition framework. The framework has been fully implemented for the service choreography simulated with inter-process communication among participating partners listening to an event-driven message broker. The decomposed local specifications are deployed to each partner as a thread running on a machine with Intel Core2 Duo 1.80GHz processors and 2GB of RAM. A global process is randomly generated as a directed graph with a varying number of tasks, partners, and task executions. Each task specifies a pair of sender and receiver and a message to be exchanged. Every edge denotes a transition between two tasks. Given the set of partners whose initial task is to send a message,

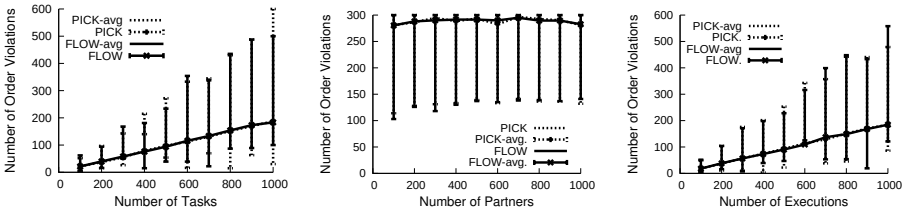


Fig. 5. Number of ordering violations.

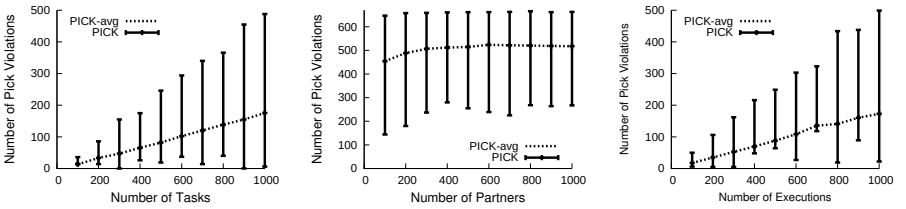


Fig. 6. Number of pick violations.

the message broker randomly chooses an initial sender and triggers it at a constant rate to execute the sending task during every simulation. The architecture of the underlying messaging substrate is orthogonal to our problem definition as long as the messaging substrate can deliver messages in order across different senders and receivers. However, the centralized message broker we assume is subject to a single point of failure. Also, synchronous handling of the coordination messages may delay highly concurrent processes executing on the same broker. Therefore, the strategy to run a multi-broker-based messaging system, e.g., [15], in an efficient and fault-tolerant manner is an interesting future work direction.

In the following subsections, we empirically assess how much the service choreography based on the *conventional* decomposition framework method be prone to produce message communication patterns that do not conform to the global specification. Specifically, we measure the number of messages delivered in out-of-order and the number of pick violations that reflect the degree of potential semantic conflicts. Then we measure the overhead of the new approach to avoid such potential semantic conflicts in terms of the number of coordination messages to be sent, and evaluate the runtime cost to execute the decomposition framework. Also we measure the increase of local tasks along with the size of the rules the message broker has to maintain in order to monitor invalid message patterns during runtime.

5.1 Measurements of Conflicts

Figure 5 and Figure 6 indicate that the number of violations of the ordering and pick constraints grow proportionally to the number of tasks and the number of executions. The number of violations is not affected by the number of peers, as the constraints on

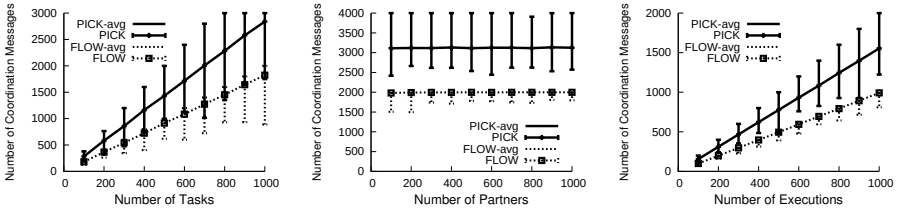


Fig. 7. Number of coordination messages exchanged among partners.

the message communication sequence from the global perspective are agnostic of the participating peers, *i.e.*, no matter how many different partners are involved in the process, ordering requirements must be met. About 20% of among 2,000 task executions violated the constraints. The results show high variance in the degree of the violations. Identifying the structural properties that make a global specification prone to the violations is subject to a future work. The fact that different partners contribute to the violations at different times and locations makes it a complicated and costly task to identify the root cause of the violation and to roll back the tasks already executed. The significant amount of violations reflects that detrimental damage already has been done to the business collaboration through the unreliable service choreography. The framework we proposed in Section 4 prevents the costly damage control with reasonable cost as shown in the following section.

5.2 Overhead of Execution

The conflict-free service choreography is realized at the cost of strictly coordinating the messages exchanged among the partners when necessary as implied in the global specification. Figure 7 shows that the number of coordination messages increases with the number of tasks and the number of executions, both for the pick and flow patterns. Pick patterns yielded more coordination messages than the flow patterns, since not only a coordination message has to be exchanged between two consecutive global tasks if necessary, but also a coordination message plus the confirmation from the receivers at the leaf tasks have to be sent to the broker following the pick operation. In our experiment each control message simply consists of a 1 byte character to distinguish the type of message, and a 4 byte integer to identify the message. Thus, for 2,000 executions, total of only 14KB on average are used by the system. In practice, the control message does not have to be significantly larger than ours.

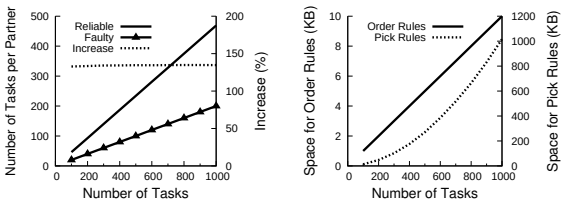


Fig. 8. Task increase per partner and space requirement for coordinator

Figure 8(b) shows the size of the rules that are used to detect conflicts by the broker in production. About 10KB and 1MB are needed for the rules to detect ordering and pick violations, respectively, given 1,000 tasks and 10 partners. Figure 8(a) shows that each partner may experience about 130% increase in the number of tasks in its local specification, as the tasks of either sending or receiving coordination messages are newly added through the new decomposition method.

5.3 Overhead of Decomposition

We measured the elapsed time to decompose a service choreography process. Given a service choreography process with 1,000 tasks and 10 partners, it required 30% to 38% more time on average to decompose the process in a pick and a flow pattern, as shown on Figure 9. With the varying number of partners, up to 1,000, our decomposition method took 72% and 56% more time on average for a pick and a flow pattern, respectively. The general increase exhibited through out experiment is due to the additional arrangement of the coordination among partners. Yet, on average the elapsed time still remains within submilliseconds, thus our method does not cost significantly more than the conventional decomposition method, in terms of computation.

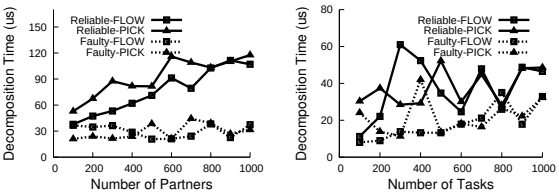


Fig. 9. Decomposition runtime.

6 Discussion

In our solution, we enrich the global specification with extra coordination messages to make it realizable. Such coordination messages may not exactly reflect the design intention of the global specification and thus restrict the behavior of services, but our solution discovers the hidden and missing synchronization among the service conversations. Choreography designers can check whether the hidden and missing synchronization is design faults and choose to fix the problems using our recommended solution or other solutions.

We assume that the decomposed local specifications are deployed to the partners offline rather than in production. If deployed during runtime of a collaboration, the newly decomposed specification can co-exist with parts of the old specification, which inevitably causes semantic conflicts, and remains uncaught by our current framework for detecting specification violations. To account for such a scenario of evolution of global specifications, our framework must include features to efficiently roll back conflicting tasks by causality tracking methods to pinpoint the source of the problem.

7 Related Work

In this section, we review the recent works on service composition, particularly, verification and design methodology.

Realizability Issue.In service choreography, local specifications govern the behavior of each service to guarantee that their interaction conforms to the global specification. An assumption underlying such an approach is that the global specification should be realizable, in the sense that the global specification can be decomposed into a set of local specifications whose behavior conforms to the global specification. However, verifying the realizability of global specification is undecidable in general [4, 13].

Much work has been done to develop sufficient conditions to determine the realizability or weak realizability of specifications. Fu *et al.* [8, 13] proved that synchronizability is a sufficient condition for realizability in the context of asynchronous service conversations. Alur *et al.* [4] studied the realizability issue for bounded message sequence charts. Two kinds of realizability relationships are studied, namely *weak realizability* and *safe realizability*, depending on whether the distributed local specifications are deadlock-free or not. Abadi *et al.* [3] studied the realizability of specifications and how to check them in reactive systems. Ben-Abdallah *et al.* [6] proposed an approach to detect problems in message sequence charts: (1) process divergence; and (2) non-local choice. Contrast to these verification works, we rather *repair* a possibly incomplete specifications by adding extra coordination information to make the resulting specification realizable at a reasonable cost. The purpose is to enforce stronger constraints into the local specifications to prevent semantic conflicts in the service choreography.

Conformance Checking. Another important issue in service choreography is to check whether a realizable global specification is correctly implemented by some local specifications. This is usually called conformance checking. For example, Aalst *et al.* [2] proposed an approach to evaluate conformance relationships between the actual service behavior at runtime and its global specification. *fitness* and *appropriateness* are proposed to measure how well the runtime behavior matches the global specification and how concise the global specification is with respect to the runtime behavior. Montali *et al.* [18] proposed to describe the global specification with a declarative language and verify the services' behavior based on logic. Fu *et al.* [12] proposed to use automata to verify service compositions with asynchronous communication. Foster *et al.* [11] proposed to verify the safety and liveness properties of service compositions based on process algebra. However, these approaches do not address how to fix possible conformance violations, which may be caused by a non-realizable global specification. Baudru *et al.* [5] proposed an implementation of regular message sequence chart specifications to avoid deadlocks with additional message contents. In contrast, our approach contributes to solve the problem by discovering and revising the problematic global specification so that it can be decomposed into compatible local specifications.

Decomposition Algorithms. Many work has also been proposed to design and compose Web services using a top-down approach. In such an approach, service developers develop a global specification (e.g., a WS-CDL [22] document) and decompose it into local specifications. Services are then selected to implement such local specifications. Aalst *et al.* [1] proposed an approach to decompose global specifications into local public views using petri-nets. Four kinds of consistency relationships can be spec-

ified between these public views and the global specification based on the application requirements. Broy *et al.* [7] proposed a formal model to split a service into sub-services by projection. Giese *et al.* [14] proposed a modular approach to design, decompose, refine and verify complex systems. Nanda *et al.* [19] proposed an approach to partition a composite Web service written as a single BPEL program into an equivalent set of decentralized processes. Li *et al.* [15] proposed a framework to deploy a centralized BPEL process into a set of distributed execution engines. An assumption underlying these approaches is that the global specification (or BPEL process) can be realized in a distributed implementation. In our work, we also provide a set of algorithms to decompose a global specification into a set of local ones. The difference is that we enrich the problematic global specification with extra coordination information to make the problematic global specification realizable. Therefore, our algorithms can be applied to a richer set of scenarios.

Specification Synthesis. In the software engineering literature, many studies have been conducted to synthesize specifications from partial and incomplete scenarios. Uchitel *et al.* [20, 21] proposed a model to synthesize partial scenarios into a global specification by interactively refining the global specification with discovered implicit scenarios. Canal *et al.* [9] proposed a model-based approach to adapt component behavior to avoid component mismatching (e.g., deadlock). Castejon *et al.* [10] proposed to synthesize realizable specifications from realizable sub-specifications by following guidelines for each synthesis operator. Our work also synthesizes the existing problematic global specification with extra coordination information, but not from local specifications.

8 Conclusions

This paper introduces a novel framework to derive a realizable global specification that guarantees to control a distributed collaboration in a complete and consistent manner during runtime. Our framework extracts implicit coordination constraints in a given global specification. By incorporating the constraints explicitly into local specifications of the collaboration partners, the overall behavior of the service choreography conforms to the global specification. The types of potential semantic conflicts arise from non-realizable global specifications. Such conflicts are modeled in the Pi-Calculus and proven to be prevented by our coordination techniques. The enrichment of the global specification is fully automated, without putting any extra burden for the collaboration designer to go refactor the global specification manually. Lastly, our comprehensive evaluation with the extensive simulation of the service choreography shows that the framework incurs a tolerable amount of additional overhead in terms of space, computation, and network usage.

References

1. W. M. P. d. Aalst and T. Basten. Inheritance of workflows: an approach to tackling problems related to change. *Theor. Comput. Sci.*, 270(1-2):125–203, uary.
2. W. M. P. v. d. Aalst, M. Dumas, C. Ouyang, A. Rozinat, and E. Verbeek. Conformance checking of service behavior. *ACM Trans. Internet Technol.*, 8(3):1–30, 2008.

3. M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable specifications of reactive systems. In *ICALP '89*, pages 1–17, London, UK, 1989. Springer-Verlag.
4. R. Alur, K. Etessami, and M. Yannakakis. Realizability and verification of msc graphs. *Theor. Comput. Sci.*, 331(1):97–114, 2005.
5. N. Baudru and R. Morin. Safe implementability of regular message sequence chart specifications. In *SNPD'03*, pages 210–217, 2003.
6. H. Ben-Abdallah and S. Leue. Syntactic detection of process divergence and non-local choice in message sequence charts. In *TACAS '97*, pages 259–274, London, UK, 1997. Springer-Verlag.
7. M. Broy, I. H. Krüger, and M. Meisinger. A formal model of services. *ACM Trans. Softw. Eng. Methodol.*, 16(1):5, 2007.
8. T. Bultan and X. Fu. Specification of realizable service conversations using collaboration diagrams. In *SOCA '07*, pages 122–132, Washington, DC, USA, 2007. IEEE Computer Society.
9. C. Canal, P. Poizat, and G. Salaün. Model-based adaptation of behavioral mismatching components. *IEEE Trans. Softw. Eng.*, 34(4):546–563, 2008.
10. H. N. Castejon, R. Braek, and G. von Bochmann. Realizability of collaboration-based service specifications. In *APSEC '07*, pages 73–80, Washington, DC, USA, 2007. IEEE Computer Society.
11. H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *ASE '03*, pages 152–161, Los Alamitos, CA, USA, 2003.
12. X. Fu, T. Bultan, and J. Su. Analysis of interacting bpel web services. In *WWW '04*, pages 621–630, New York, NY, USA, 2004. ACM.
13. X. Fu, T. Bultan, and J. Su. Synchronizability of conversations among web services. *IEEE Trans. Softw. Eng.*, 31(12):1042–1055, 2005.
14. H. Giese, S. Burmester, W. Schäfer, and O. Oberschelp. Modular design and verification of component-based mechatronic systems with online-reconfiguration. In *SIGSOFT '04/FSE-12*, pages 179–188, New York, NY, USA, 2004. ACM.
15. G. Li, V. Muthusamy, and H.-A. Jacobsen. A distributed service-oriented architecture for business process execution. *ACM Trans. Web*, 4(1):1–33, 2010.
16. L. T. Ly, S. Rinderle, and P. Dadam. Integration and verification of semantic constraints in adaptive process management systems. *Data Knowl. Eng.*, 64(1):3–23, 2008.
17. R. Milner. *Communicating and mobile systems: the pi-calculus*. Cambridge University Press, New York, NY, USA, 1999.
18. M. Montali, M. Pesic, W. M. P. v. d. Aalst, F. Chesani, P. Mello, and S. Storari. Declarative specification and verification of service choreographiess. *ACM Trans. Web*, 4(1):1–62, 2010.
19. M. G. Nanda, S. Chandra, and V. Sarkar. Decentralizing execution of composite web services. In *OOPSLA '04*, pages 170–187, New York, NY, USA, 2004. ACM.
20. S. Uchitel, J. Kramer, and J. Magee. Synthesis of behavioral models from scenarios. *IEEE Trans. Softw. Eng.*, 29(2):99–115, 2003.
21. S. Uchitel, J. Kramer, and J. Magee. Incremental elaboration of scenario-based specifications and behavior models using implied scenarios. *ACM Trans. Softw. Eng. Methodol.*, 13(1):37–85, 2004.
22. W3C. Web services choreography description language version 1.0, 2005. http://www.w3.org/TR/ws-cdl-10.
23. Y. Yoon, C. Ye, and H.-A. Jacobsen. On semantics conflict in service choreography. In *Middleware Systems Research Group Technical Report*, Toronto, Canada, March 2010.