

Received May 14, 2019, accepted June 17, 2019, date of publication July 1, 2019, date of current version July 17, 2019.

Digital Object Identifier 10.1109/ACCESS.2019.2926097

A Correctness Enforcement Approach for Collaborative Business Processes

QI MO^{1,2}, LIRUI BAI^{1,4}, FEI DAI^{2,3}, JIANGLONG QIN^{1,2}, ZHONGWEN XIE^{1,2}, AND TONG LI^{1,2}

¹School of Software, Yunnan University, Kunming 650091 China

²Key Laboratory of Software Engineering of Yunnan Province, Kunming 650091, China

³School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming 650091, China

⁴Department of Computer Science and Technology, Tianjin University Renai College, Tianjin 301636, China

Corresponding authors: Lirui Bai (bai_lirui@126.com), Fei Dai (59671019@qq.com), and Jianglong Qin (83803036@qq.com)

This work was supported in part by the Project of National Natural Science Foundation of China under Grant 61862065, Grant 61702442, and Grant 61662085, in part by the Application Basic Research Project in Yunnan Province under Grant 2018FB105, in part by the Open Foundation of Key Laboratory for Software Engineering of Yunnan Province under Grant 2017SE201 and Grant 2016SE202, and in part by the Yunnan Province Young Academic and Technical Leaders Funds for Training under Grant C6143002.

ABSTRACT Collaborative business processes are indeed complex and difficult to be correctly designed as business processes involved in them are typically developed by different participating organizations and there is no way to foresee all potential interactions at design time. To this end, we propose a novel correctness enforcement approach for collaborative business processes. In this approach, given an original process, we first propose an algorithm for automatically detecting its correctness. Then, in case the original process is partially correct, we prune it to generate its core. The core captures all legal traces in the original process. Finally, an enforced process is yielded from the core through coordination mapping (i.e., inserting additional coordination activities into original processes). Our approach is implemented as a prototype tool called cet (a correctness enforcement tool for collaborative business processes), and a series of experiments is conducted to validate the applicability and effectiveness of the proposed approach.

INDEX TERMS Collaborative business process, correctness, core, minimization, coordination mapping.

I. INTRODUCTION

Business processes are the main asset of enterprises as they describe their value chain and fundamentally define the “way, businesses are done” [1]–[3]. Unlike traditional business processes that are limited to a single organization, collaborative business processes cross boundaries of organizations, and have the capability of gathering a set of business processes with complementary competencies and knowledge to cooperate to achieve more business successes [4]–[5]. Currently, they have been applied in multiple industries. Some prominent examples are enterprise information systems based on PAIS (Process-Aware Information Systems) [6], e-commerce business processes [7], [8], and medical business processes [9].

Yet, business processes gathered in the collaborative business process are developed independently by different organizations. Due to autonomy reasons, i.e., each participating organization acts independently and is not obliged to reveal its own private information (or process) to other parties, there

is no way to foresee all potential interactions between them. As a consequence, some undesirable outcomes such as deadlocks and livelocks may occur during their actual cooperation.

To avoid these undesirable outcomes, existing approaches mainly focus on *correctness checking*. Given a correctness criterion such as soundness or weak termination [4], correctness checking approaches automatically detect the correctness of a collaborative business process using formal verification techniques. If the result of the detection is incorrect, then diagnosis information can be used to repair the collaborative business process. As presented in Figure 1(a), correctness can be eventually reached through iterative detection and adjustment for the collaborative business process.

A more promising approach is *correctness enforcement*. As presented in Figure 1(b), the approach focuses on the early design phase and uses a collaborative business process (called an original process) and a correctness criterion (e.g., soundness) as input. If the original process is fully incorrect or correct, then there is no need to enforce it [10]–[13]. However, if the original process is partially correct, the approach can automatically generate a collaborative business process that is correct, i.e., an enforced

The associate editor coordinating the review of this manuscript and approving it for publication was Aakash Ahmad.

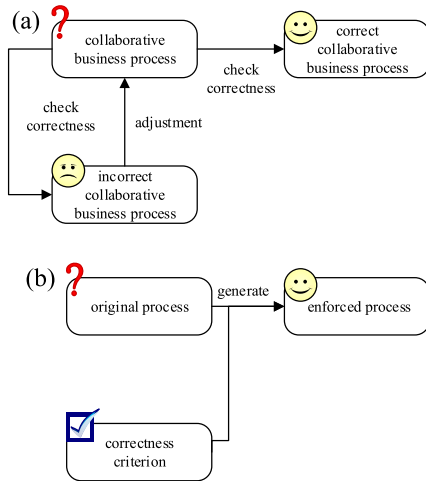


FIGURE 1. Approaches to achieve correctness. (a) Correctness checking, and (b) correctness enforcement.

process. Compared with the approach for *correctness checking*, the approach presents two advantages. First, iterative detection and adjustment are not required, which will greatly improve the modeling efficiency of collaborative business processes. Second, correctness is not only detected, but actually enforced.

In this paper, we investigate the later approach. Yet, existing correctness enforcement approaches (e.g., [10]–[13]) need to put additional restrictions on original processes when achieving correctness enforcement. Additionally, they mainly focus on *asynchronous communication* (i.e., message-based asynchronous interactions between business processes), but fail to consider *synchronous communication*. Like asynchronous communication, synchronous communication is an important interactive way as well, which forces multiple business processes to perform a specific activity at the same time [4]. In practical applications, these approaches suffer from the applicability problem as the two problems are not well addressed. The applicability means that given any original process that is partially correct, an enforced process can be generated eventually.

To this end, we propose a novel approach for enforcing collaborative business processes. Our approach aims to solve the applicability problem in existing approaches (e.g., [10]–[13]) in the case of ensuring the effectiveness. The effectiveness means that the generated enforced process is correct and maximally permissive. The term “maximally permissive” means that the generated enforced process preserves all legal traces in its original process without introducing additional traces in the case of neglecting coordination factors. Note that coordination factors in our context correspond to a class of special activities that are synchronized to achieve coordination logic.

As presented in Figure 2, our approach consists of three steps. At first, given an original process, we propose an algorithm for automatically detecting its correctness. Afterwards, in case of partially correct, we prune the original process to

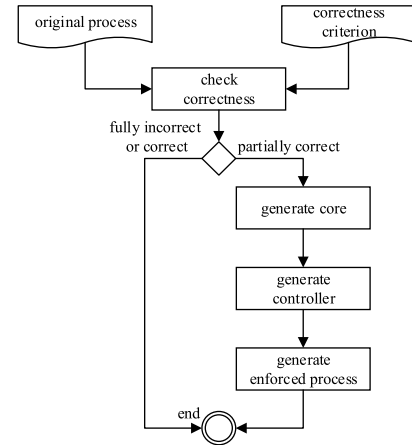


FIGURE 2. Overview of our approach.

obtain its core. At last, an enforced process can be eventually generated from the core through coordination mapping. Note that in our approach, cores can be used to capture all legal traces in original processes, while coordination mapping inserts coordination factors into original processes to ensure the fact that the generated enforced processes are correct and maximally permissive (see Definition 12).

It is worth noting that since our approach needs to build the reachability graph of original processes when achieving correctness enforcement, it may suffer from the state-space explosion problem. The problem will be alleviated in our future work.

Compared with existing approaches for correctness enforcement, the main contributions of this paper are as follows:

- (1) We propose a general approach to enforce original processes that may involve both asynchronous and synchronous communication without any restrictions. The approach can be used to solve the applicability problem in existing approaches.
- (2) We prove that enforced processes generated by our approach are effective. That is, they are correct and maximally permissive.
- (3) We implement our approach as a prototype tool called *cet*. We employ the tool to conduct experiments with 30 real-world cases. The experimental results demonstrate the applicability and effectiveness of our approach.

The remainder of this paper is organized as follows. Section 2 gives a motivating example used throughout the paper to illustrate our approach. Section 3 presents some formal definitions related to our correctness enforcement approach. Section 4 illustrates our correctness enforcement approach in detail. Section 5 presents a prototype tool implementing our approach and evaluates the applicability and effectiveness of our approach using actual cases. Section 6 discusses the related work. Section 7 concludes this paper.

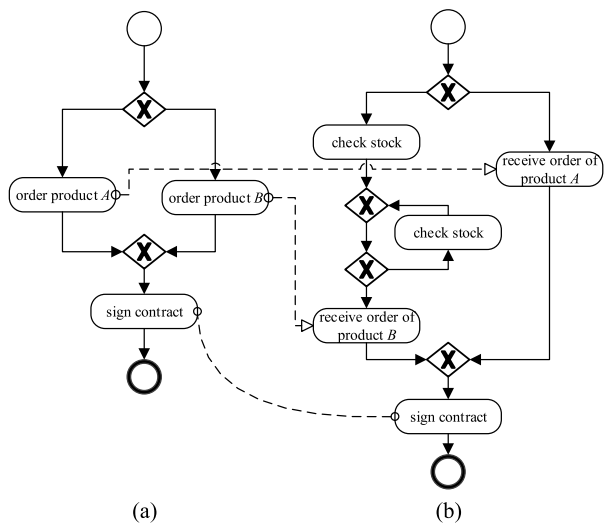


FIGURE 3. Ord depicted by BPMN.

II. MOTIVATING EXAMPLE

In this section, we present a simplified ordering process (Ord) used throughout the paper to illustrate our approach. Ord involves 2 participants, i.e., a customer (denoted as *Cus*) and a vendor (denoted as *Ven*). Their business processes and the interaction between them are depicted in Figure 3.

Concretely, the interaction includes the following steps:

- (1) *Cus* can order product A or B from *Ven*;
- (2) The inventory of product A is sufficient in any case, and therefore *Ven* directly receives the request for ordering product A. Yet, since the inventory of product B is usually insufficient, *Ven* needs to first check its stock before receiving the request for ordering product B;
- (3) After receiving the request for ordering product A or B, *Cus* and *Ven* sign a contract to complete the ordering process.

In practice, the smooth interaction between *Cus* and *Ven* is critical to the correctness of *Ord*. In the scenario shown in Figure 1, we can see that 1) ordering product A or B is done through the exchange of asynchronous messages, but the contract requires *Cus* and *Ven* to sign at the same time. Hence, both synchronous and asynchronous communication are involved in *Ord*; and 2) despite the fact that the messages exchanged in *Ord* are consistent, *Ord*'s actual execution may produce undesirable outcomes such as livelocks and unspecified receptions. For example, if *Cus* first orders a product A. Yet, *Ven* first checks its inventory, and then receives *Cus*'s order. This will cause a livelock and an unspecified reception during the execution of *Ord*.

To eliminate these undesirable outcomes, we propose a novel approach for correctness enforcement. Our approach is suitable for the enforcement of complex cases which involve both synchronous and asynchronous communication. Additionally, it does not require adding any additional restrictions on the original process when achieving correctness enforcement.

III. PRELIMINARIES

In this section, we first present several definitions related to Labeled Transition Systems (LTSs). Then, we model collaborative business processes using LTSs. Finally, we define the correctness of collaborative business processes using weak termination.

A. LABELED TRANSITION SYSTEMS

Since we use LTSs to specify some notions (e.g., original processes, enforced processes and cores) involved in our correctness enforcement approach, several definitions related to Labeled Transition Systems are presented in the following.

Definition 1 (Labeled Transition System): A labeled transition system is a tuple $LTS = (S, A, T, s_0, F)$, where

- (1) S is a set of states;
- (2) $s_0 \in S$ is the initial state;
- (3) $F \subseteq S$ is a set of final states;
- (4) A is a set of activities;
- (5) $T \subseteq S \times A \times S$ is a transition relation.

For the sake of simplicity, we write $r \xrightarrow{a} s$ to denote $(r, a, s) \in T$.

Definition 2 (Reachability): Let $LTS = (S, A, T, s_0, F)$ be an LTS, and s and s' be two states in S . s' is called reachable from s (denoted as $s \xrightarrow{*} s'$), iff there exists a sequence of labels (i.e., a trace) $\sigma = a_1 \dots a_n$, such that $s \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s'$. Particularly, if $s = s_0$ and $s' \in F$, then the trace is called a legal trace. Note that the empty trace is also allowed, i.e., $s \xrightarrow{*} s$.

Based on the reachability, we can define transition closures. Formally, a transition closure is defined as follows.

Definition 3 (Transition Closure): Let $LTS = (S, A, T, s_0, F)$ be an LTS, and $s \in S$ be a state, then the transition closure of s is defined as $t\text{-closure}(s) = \{s' \mid s \xrightarrow{*} s'\}$.

In essence, the transition closure of a state refers to a set of states reachable from the state. Transition closures will be used to detect the correctness of the original process and generate cores in Section 4.

B. COLLABORATIVE BUSINESS PROCESSES

Based on business processes, we can model collaborative business processes. A business process which consists of activities and their control flow usually pertains to a specific organization. In our context, we use LTSs to describe business processes.

In essence, LTSs are a low-level formal model yet business processes in practice are typically described by Business Process Modeling Notation (BPMN). Currently, several approaches have been presented to convert them into process models described by Petri nets and process algebras [24], [29]. Since Petri nets and process algebras have well-defined execution semantics, these process models can be transformed into LTSs in an easy way.

To reduce the difficulty of directly describing the business process using LTSs (this is mainly caused by the parallelism of activities), in this paper we use Finite State Process (FSP) [17]. Technically, FSP is a process calculus designed to be

easily machine and human readable that includes standard constructs such as action prefix (\rightarrow), external choice ($|$), hiding (\backslash), relabeling ($/$) and parallel composition ($||$), which can be used to specify LTSs in a textual representation [17].

Definition 4 (Business Process): A business process is an LTS $BP = (S, A, T, s_0, F)$, where

- (1) S is a set of states;
- (2) $s_0 \in S$ is the initial state;
- (3) $F \subseteq S$ is a set of final states;
- (4) $A = A_l \cup A_i$ is a set of activities, where A_l is a set of local activities, and A_i is a set of interactive activities;
- (5) $A_i = A_{sy} \cup A_{ay}$, where A_{sy} is a set of synchronous interactive activities, and A_{ay} is a set of asynchronous interactive activities;
- (6) $A_{ay} = A_s \cup A_r$, where A_s is a set of asynchronous sending activities, and A_r is a set of asynchronous receiving activities;
- (7) $T \subseteq S \times A \times S$ is a transition relation, which specifies sequential constraints between activities, i.e., the control flow.

In our context, for an asynchronous sending activity a , a is written as $!m$, where m is the message name and “!” stands for emission. Analogously, for an asynchronous receiving activity a , a is written as $?m$, where m is the message name and “?” stands for reception. For the sake of simplicity, we used $M_s = \{m|!m \in A_s\}$ to denote the messages sent by BP , and $M_r = \{m|?m \in A_r\}$ to denote the messages received by BP .

Recall that in this paper, we consider both synchronous and asynchronous communication. In the latter case, each business process in the collaboration is equipped with a FIFO queue for storing messages received from other business processes, and from which the current business process can consume messages. In particular, in this paper we don't limit the length of message queues and assume that they are unbounded.

Example 1: According to the scenario depicted in Figure 3, we first describe the business processes of *Cus* and *Ven* using FSP as follows.

```

/*****FSP Coding of Cus*****/
Cus = !orderA→C1 | !orderB→C1;
C1 = signContract→END.
/*****FSP Coding of Ven*****/
Ven = checkStock→V1 | ?orderA→V2;
V1 = checkStock→V1 | ?orderB→V2;
V2 = signContract→END.
    
```

Then, we generate their LTSs based on the operational semantics of FSP [17], as shown in Figure 4.

Business processes can be composed to build collaborative business processes. Given a set of business processes, their composition can be formally defined as follows.

Definition 5 (Composition): Let BP_1, \dots, BP_n be a set of business processes. For any $i \in [1..n]$: $BP_i = (S_i, A_i, T_i, s_{i0}, F_i)$, and Q_i is its associated message queue. Then, their composition can be defined as an LTS $BP_1 || \dots || BP_n = (S, A, T, s_0, F)$, where

- (1) $S \subseteq S_1 \times Q_1 \times \dots \times S_n \times Q_n$ is a set of states;

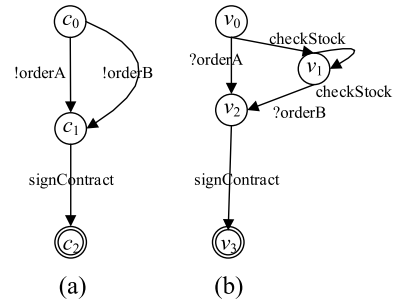


FIGURE 4. Business processes of Cus and Ven. (a) Business process N_1 of Cus and (b) business process N_2 of Ven.

- (2) $s_0 = \langle s_{10}, \epsilon, \dots, s_{n0}, \epsilon \rangle \in S$ is the initial state, and ϵ indicates that the message queue is empty;
- (3) $F \subseteq S$ are final states, and for each final state $s_f \in F$, $s_f = \langle s_{f1}, \epsilon, \dots, s_{fn}, \epsilon \rangle$, where $\forall i \in [1..n]$, s_{fi} is a final state of BP_i , and Q_i is empty;
- (4) $A \subseteq A_1 \cup \dots \cup A_n$ is a set of activities;
- (5) $T \subseteq S \times A \times S$ is a transition relation, and for two states $s = (s_1, Q_1, \dots, s_n, Q_n)$ and $s' = (s'_1, Q'_1, \dots, s'_n, Q'_n)$
 - (a) $\exists i \in [1..n]$: $a \in BP_i.A_l$ and $s \xrightarrow{a} s' \in T$ if $s_i \xrightarrow{a} s'_i \in BP_i.T, \forall k \in [1..n]: Q'_k = Q_k, s'_k = s_k (k \neq i)$, and $s'_k = s_i (k = i)$;
 - (b) $\exists i \in [1..n]$: $a \in BP_i.A_{sy}$ and $s \xrightarrow{a} s' \in T$ if $\forall j \in [1..n]$ and $a \in BP_j.A_{sy}: s_j \xrightarrow{a} s'_j, \forall k \in [1..n]: Q'_k = Q_k, s'_k = s_k (k \neq j)$, and $s'_k = s'_j (k = j)$;
 - (c) $\exists i \in [1..n]$: $a = !m \in BP_i.A_s$ and $s \xrightarrow{a} s' \in T$ if $s_i \xrightarrow{a} s'_i \in BP_i.T, \exists i, j \in [1..n]: m \in M_{is} \cap M_{jr}, \forall k \in [1..n]: Q'_k = Q_k (k \neq j), Q'_k = Q_k m (k = j), s'_k = s_k (k \neq i)$, and $s'_k = s_i (k = i)$;
 - (d) $\exists i \in [1..n]$: $a = ?m \in BP_i.A_r$ and $s \xrightarrow{a} s' \in T$ if $s_i \xrightarrow{a} s'_i \in BP_i.T, Q_i = mQ_i'', \forall k \in [1..n]: Q'_k = Q_k (k \neq i), Q'_k = Q_k'' (k = i), s'_k = s_k (k \neq i)$, and $s'_k = s_i (k = i)$.

In Definition 5, (a) defines the execution of a local activity. (b) defines synchronous communication between business processes, which force multiple business processes to execute activity a at the same time. (c) and (d) together define asynchronous communication between business processes. Concretely, (c) describes the execution of an asynchronous sending activity a , it requires that there exists a business process (i.e., BP_j) that can receive m . After m is sent, m is added to the tail of message queue Q_i , while (d) describes the execution of an asynchronous receiving activity a in BP_i , it requires that the message queue Q_i is not empty and the message stored in the head of Q_i is m . After m is received, m is removed from the head of Q_i .

Based on Definition 5, we know that both synchronous and asynchronous communication are defined during the composition of business processes. Hence, our approach can model collaborative business processes that involve both asynchronous and synchronous communication.

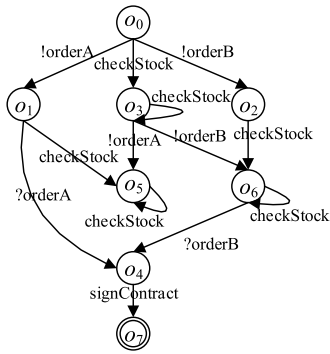
FIGURE 5. The original process OP .

TABLE 1. The information description of states.

State	Description	State	Description
o_0	$(c_0, \epsilon, v_0, \epsilon)$	o_4	$(c_1, \epsilon, v_2, \epsilon)$
o_1	$(c_1, \epsilon, v_0, [\text{orderA}])$	o_5	$(c_1, \epsilon, c_1, [\text{orderA}])$
o_2	$(c_1, \epsilon, v_0, [\text{orderB}])$	o_6	$(c_1, \epsilon, v_1, [\text{orderB}])$
o_3	$(c_0, \epsilon, v_1, \epsilon)$	o_7	$(c_2, \epsilon, v_3, \epsilon)$

Support that BP_1, \dots, BP_n are business processes in the collaboration, then the collaborative business process built by composing these business processes can be defined as $BP_1 || \dots || BP_n$.

Example 2: By composing of N_1 and N_2 depicted in Figure 4, we can construct the original process OP of Ord , as shown in Figure 5.

Specifically, the information description of the states in OP is presented in Table 1, where orderA and orderB are messages generated during the collaboration, while c_0 - c_2 and v_0 - v_3 are the states in N_1 and N_2 , respectively.

Given a collaborative business process $CBP = (S, A, T, s_0, F)$, if S is infinite, then CBP is called unbounded, otherwise CBP is bounded. In our context, we will restrict ourselves to bounded collaborative business processes.

Example 3: In Figure 5, we can see that OP has eight states, and therefore it is bounded.

C. CORRECTNESS

Currently, multiple criteria for defining the correctness of collaborative business processes have been presented. Among them, soundness [4] and its variants (such as weak termination [11]) are widely used. Typically, soundness is used to define the correctness of intra-organizational business processes, and it is too strict for cross-organizational business processes [11]. Hence, in this paper we define the correctness of collaborative business processes using weak termination.

Definition 6 (Correctness): Let $CBP = (S, A, T, s_0, F)$ be a collaborative business process, CBP is correct iff for each state s with $s_0 \xrightarrow{*} s$, there exists a final state $s_f \in F$, such that $s \xrightarrow{*} s_f$.

Definition 6 requires that a final state is always reachable from every reachable state. This ensures that each business process can reach one of its final states. Therefore, any

deadlocks and livelocks in the collaborative business process are removed. Additionally, since each message queue in the final state is empty (see Definition 5), ensuring that all messages generated in actual collaboration are reasonably received.

In Particular, given a collaborative business process $CBP = (S, A, T, s_0, F)$, if all states in S can not reach final states, then CBP is called incorrect. If some but not all states in S can reach final states, then CBP is called partially correct.

IV. CORRECTNESS ENFORCEMENT APPROACH

In this section, we first give an algorithm to detect the correctness of original processes. Then, in case original processes are partially correct, we present an algorithm to obtain their cores. Finally, we present an approach to generate enforced processes from cores through coordination mapping.

A. CORRECTNESS DETECTION

To achieve correctness enforcement, we need first determine the correctness of original processes.

Based on transition closures, we propose an algorithm to detect the correctness of original processes.

Algorithm 1 Detect the Correctness of Original Processes

Input: An original process $OP = (S, A, T, s_0, F)$;

Output: “Incorrect”, “Partially correct”, or “Correct”;

1. Set $validStates = \emptyset$;
2. **for each** state s in S **do**
3. generate t -closure(s);
4. **if** t -closure(s) $\cap F \neq \emptyset$ **then**
5. add s into $validStates$;
6. **end if**
7. **end for**
8. **if** $|validStates| = |S|$ **then**
9. return “Correct”;
10. **else if** $|validStates| \neq 0 \wedge |validStates| < |S|$ **then**
11. return “Partially correct”;
12. **else**
13. return “Incorrect”;
14. **end if**

For each state $s \in S$, Algorithm 1 first generates its transition closure ($L1 \sim L7$). If all states in S can reach final states, then Algorithm 1 outputs “Fully correct”, meaning that the original process is correct ($L8 \sim L9$). If some but not all states in S can reach final states, then Algorithm 1 outputs “Partially correct”, meaning that the original process is partially correct ($L10 \sim L11$). If all states in S can not reach final states, then Algorithm 1 outputs “Incorrect”, meaning that the original process is incorrect ($L12 \sim L13$).

Recall that we focus in this paper on bounded collaborative business processes. Therefore, we know that the states in S are finite, and then derive that Algorithm 1 can be terminated. Assuming that $n = |S|$, then the time complexity

of Algorithm 1 is $O(n^2)$, where $|S|$ indicates the number of the states in S .

Example 4: For the original process OP in Figure 5, according to Definition 3, we first calculate the transition closure of each state in OP as follows:

- $t\text{-closure}(O_0) = \{O_0, \dots, O_7\}$;
- $t\text{-closure}(O_1) = \{O_1, O_4, O_5, O_7\}$;
- $t\text{-closure}(O_2) = \{O_2, O_4, O_6, O_7\}$;
- $t\text{-closure}(O_3) = \{O_3, O_5\}$;
- $t\text{-closure}(O_4) = \{O_4, O_7\}$;
- $t\text{-closure}(O_5) = \{O_5\}$;
- $t\text{-closure}(O_6) = \{O_4, O_6, O_7\}$;
- $t\text{-closure}(O_7) = \{O_7\}$.

Then, based on Algorithm 1, we can derive that OP is partially correct as some but not all states in OP can reach the final state O_7 .

B. CORE GENERATION

In case original processes are partially correct, we can prune them to obtain their cores. In essence, cores capture all legal traces in original processes. In our context, they are described using LTSs and can be used to generating enforced processes in Section 4.3.

Based on transition closures, we propose a novel algorithm to generate a core from an original process.

Algorithm 2 Generate the Core

Input: An original process $OP = (S, A, T, s_0, F)$;

Output: A core $C = (S_c, A_c, T_c, s_{c0}, F_c)$;

1. Set $invalidStates = \emptyset$;
 2. **for each** state s in S **do**
 3. generate $t\text{-closure}(s)$;
 4. **if** $t\text{-closure}(s) \cap F = \emptyset$ **then**
 5. add s to $invalidStates$;
 6. **end if**
 7. **end for**
 8. set $S_c = S - invalidStates$;
 9. **for each** transition $t = (s_1, a, s_2)$ in T **do**
 10. **if** $s_1 \in invalidStates$ or $s_2 \in invalidStates$ **then**
 11. **continue**;
 12. **end if**
 13. add t to T_c ;
 14. **end for**
 15. $s_{c0} = s_0$; $F_c = F$;
 16. **for each** transition $t = (s_1, a, s_2)$ in T_c **do**
 17. add a to A_c ;
 18. **end for**
-

For each state $s \in S$, Algorithm 2 first generates its transition closure, and determines its validity (L1~L7). Then, for each invalid state, Algorithm 2 removes it from S (L8). Finally, Algorithm 2 deletes the transitions formed by these invalid states (L9~L14). Note that a state $s \in S$ is valid if and only if there exists a trace $\sigma = a_1 \dots a_n$, such that $s \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_f$, where $s_f \in F$. That is, its transition

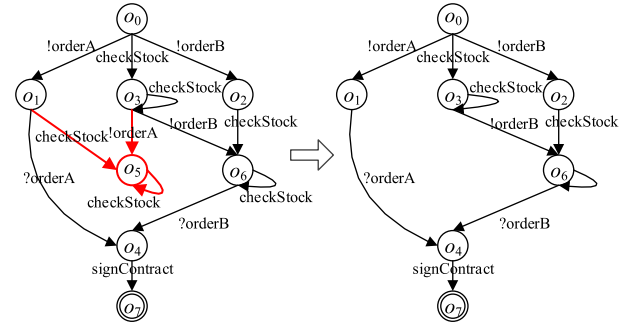


FIGURE 6. The core C of OP .

closure includes final states. Assuming that $n = |S|$, then the time complexity of Algorithm 2 is $O(n^2)$, where $|S|$ indicates the number of the states in S .

According to Algorithm 2, we can derive that Lemma 1 holds.

Lemma 1: Given an original process OP and its corresponding core C , then C preserves all legal traces in OP without introducing additional traces.

Proof: According to Algorithm 2, we know that all invalid states and the transitions formed by them are removed from OP while all valid states and the transitions formed them are preserved in C . For each valid state, it can always reach a final state. Yet, for an invalid state, it can't reach a final state. Therefore, according to Definition 2, we can derive that C preserves all legal traces in OP without introducing additional traces. \square

Example 5: For the original process OP in Figure 5, according to Algorithm 2, we can generate its core C , as shown in Figure 6.

In C , we can see that the invalid state O_5 and the transitions (marked in red) formed by it are removed from OP while ensuring that all valid states and the transitions formed them still remain in C .

C. ENFORCED PROCESSES GENERATION

Based on an obtained core, an enforced process can be generated by the following steps:

- (1) Analyze the core and determine its coordination transitions;
- (2) Based on these coordination transitions, generate a set of hidden cores;
- (3) Minimize these hidden cores to obtain a set of intermediate business processes;
- (4) Using coordination mapping, generate a set of enforced business processes from these intermediate business processes;
- (5) Through the composition of these enforced business processes, an enforced process is generated.

At first, we detail the approach for generating coordination transitions. In essence, coordination transitions refer to the transitions that may lead to undesirable outcomes (e.g., deadlocks and livelocks) during the execution of original processes.

Given an original process, in practice, not every transition in it needs to be coordinated. With the notion of core, coordination transitions are formally defined as follows.

Definition 7 (Coordination Transition): Let OP be an original process, and $C = (S_c, A_c, T_c, s_{c0}, F_c)$ be its corresponding core. A transition $s_1 \xrightarrow{a} s_2$ in C is called a coordination transition, iff $s_1 \in S_c$ and $s_2 \notin S_c$.

Based on Definition 7, we propose an algorithm to obtain coordination transitions in an original process.

Algorithm 3 Generate Coordination Transitions

Input: A core $C = (S_c, A_c, T_c, s_{c0}, F_c)$;

Output: Set $coordTrans$;

1. **for each** state s_1 in S_c **do**
 2. **if** $s_1 \xrightarrow{a} s_2$ and $s_2 \notin S_c$ **then**
 3. add $s_1 \xrightarrow{a} s_2$ to $coordTrans$;
 4. **end if**
 5. **end for**
-

Assuming that $n = |S_c|$ and $m = |T_c|$, then the time complexity of Algorithm 3 is $O(m \times n)$.

Example 6: For the core C in Figure 6, we can obtain its coordination transitions using Algorithm 3, i.e., $o_1 \xrightarrow{checkStock} o_5$, $o_3 \xrightarrow{!orderA} o_5$, and $o_5 \xrightarrow{checkStock} o_5$.

Afterwards, we present an approach to hide cores based on coordination transitions. Given a business process BP and a core C , the basic idea of hiding is to reset the transitions contained in C . That is, for a transition $t = s_1 \xrightarrow{a} s_2$ in C , if a is an activity in BP or t is a coordinated transition, then t remains unchanged, otherwise activity a is set to τ , i.e., the invisible activity.

Since multiple transitions in a core may be associated with the same activity, we need to formally define the consistency between transitions. In essence, a set of consistent transitions in an original process refer to a transition that may be executed by the original process at different times.

Definition 8 (Consistent Transition): Let $C = (S_c, A_c, T_c, s_{c0}, F_c)$ be a core, and $t_1 = (c_1, a, c_2)$, $t_2 = (c_3, b, c_4)$ be two transitions in C . t_1 and t_2 are consistent, iff they satisfy the following conditions:

- (1) $a = b$;
- (2) $\zeta(c_1) - \zeta(c_1) \cap \zeta(c_2) = \zeta(c_3) - \zeta(c_3) \cap \zeta(c_4)$;
- (3) $\zeta(c_2) - \zeta(c_1) \cap \zeta(c_2) = \zeta(c_4) - \zeta(c_3) \cap \zeta(c_4)$;

where let $s = (s_1, Q_1, \dots, s_n, Q_n)$ be a state in C , then $\zeta(s) = \{s_1, \dots, s_n\}$.

In definition 8, (1) states that the activities associated with t_1 and t_2 are identical, (2) states that the source states of t_1 and t_2 are identical, and (3) states that the target states of t_1 and t_2 are identical.

Example 7: For the core C in Figure 6, according to Definition 8, we can derive that $t_1 = o_0 \xrightarrow{checkStock} o_3$ and $t_2 = o_2 \xrightarrow{checkStock} o_6$ are consistent. Despite the fact that the activities associated with t_1 and t_3 (i.e., $o_3 \xrightarrow{checkStock} o_3$) are identical, the two transitions aren't consistent as their source

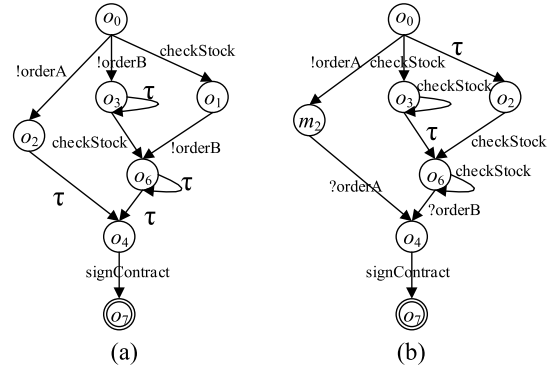


FIGURE 7. Hidden cores corresponding to Cus and Ven . (a) C_{h1} and (b) C_{h2} .

and target states aren't identified. For example, the source state of t_1 is v_0 , but the source state of t_3 is v_1 .

Definition 9 (Hiding): Let $OP = BP_1 || \dots || BP_n$ be an original process, $BP = (S, A, T, s_0, F)$ be a business process in OP , $C = (S_c, A_c, T_c, s_{c0}, F_c)$ be the core corresponding to CBP , and CT be coordination transitions in C . Then, the hidden core obtained by hiding C based on BP and CT is an LTS $C_h = (S_h, A_h, T_h, s_{h0}, F_h)$, where

- (1) $S_h = S_c$;
- (2) $A_h = \{a | \forall (r, a, s) \in CT \wedge a \in A\}$;
- (3) For each transition $t = (r, a, s)$, if $a \in A$ or $\exists ct \in CT$, ct and t are consistent, then $t \in T_h$, otherwise $(r, \tau, s) \in T_h$;
- (4) $s_{h0} = s_{c0}$;
- (5) $F_h = F_c$.

Example 8: For Cus and Ven , we can generate their hidden cores, as shown in Figure 7.

After generating hidden cores, we propose an approach to minimize them to obtain intermediate business processes. Currently, several techniques for minimizing LTSs have been presented. Among them, *weak trace minimization* [14] and *branching bisimulation minimization* [16] are widely used. Despite the fact that hidden cores in our context are LTSs, the two techniques cannot be directly used to minimize them. Weak trace minimization is too relaxed as existing indeterminate choices in hidden cores cannot be preserved. In contrast, branching bisimulation minimization is too strict as existing invisible activities in hidden cores cannot be completely removed.

To remove all invisible activities while preserving existing indeterminate choices, we propose a novel algorithm to minimize hidden cores.

In Algorithm 4, τ -closure(S) refers to the τ -closure of set S [13]. Theoretically, Algorithm 4 is closely related to weak trace minimization. The only difference between them is that weak trace minimization uses activities to generate merging states, while Algorithm 5 uses blocks to generate merging states. According to Algorithm 5, we know that transitions in each block are consistent, but transitions distributed in different blocks are inconsistent. Therefore, we can derive

Algorithm 4 Minimize the Hidden Core

Input: A hidden core $C_h = (S_h, A_h, T_h, s_{h0}, F_h)$ and coordination transitions CT ;
Output: An intermediate business process $BP_i = (S_i, A_i, T_i, s_{i0}, F_i)$;

1. add τ -closure($\{s_{h0}\}$) to empty queues Q_1 and Q_2 ;
2. **while** $Q_1 \neq \emptyset$ **do**
3. dequeue an element e from Q_1 ;
4. **for each** activity a in A_h **do**
5. obtain the transitions T based on e and a ;
6. **if** $|T| = 0$ **then**
7. **continue**;
8. **end if**
9. obtain the blocks B of T using Algorithm 5;
10. **for each** b in B **do**
11. obtain the target states S_t of transitions T_b in b ;
12. obtain τ -closure(S_t);
13. **if** τ -closure(S_t) is not in Q_2 **then**
14. add τ -closure(S_t) to Q_1 and Q_2 ;
15. generate $t = (e, a, \tau$ -closure(S_t));
16. add t to T_h ;
17. obtain a transition $t_b \in T_b$;
18. **if** $t_b \in CT$ **then**
19. set t to a coordination transition in BP_i ;
20. **end if**
21. **else**
22. generate $t = (e, a, \tau$ -closure(S_t));
23. and add t to T_h ;
24. obtain a transition $t_b \in T_b$;
25. **if** $t_b \in CT$ **then**
26. set t to a coordination transition in BP_i ;
27. **end if**
28. **end if**
29. **end for**
30. **end for**
31. **end while**
32. $S_i = Q_2$; $A_i = A_h$; $s_{i0} = \tau$ -closure($\{c_{h0}\}$);
33. **for each** state s in S_i **do**
34. **if** $s \cap F_h \neq \emptyset$ **then**
35. add s to F_i ;
36. **end if**
37. **end for**

that existing indeterminate choices are preserved. Additionally, after a merging state (such as S_t) is generated, Algorithm 4 automatically calculates its corresponding τ -closure (such as τ -closure(S_t)) as a final merging state. This ensures that all invisible activities contained in hidden cores are removed. Assuming that $m = |S_i|$, $n = |A_h|$, and $k = |B|$, then the time complexity of Algorithm 4 is $O(m \times n \times k)$.

Algorithm 5 is used to calculate blocks corresponding to a set of transitions. Assuming that $n = |T|$, then the time complexity of Algorithm 5 is $O(n^2)$.

Algorithm 5 Obtain Blocks

Input: A set of transitions T ;
Output: A set of blocks B ;

1. **for each** transition t in T **do**
2. obtain the index i of t in B ;
3. **if** $i = \emptyset$ **then**
4. new a block b and add t to b ;
5. add b to B ;
6. **else**
7. obtain the i th block b_i and add t to b_i ;
8. **end if**
9. **end for**

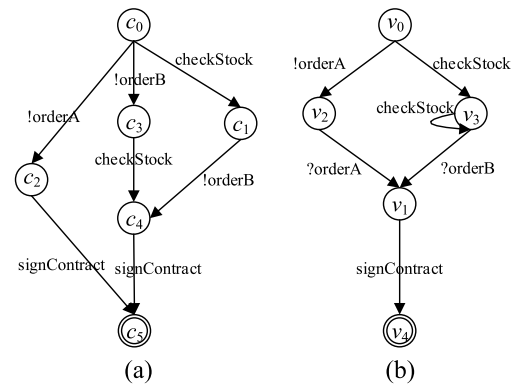


FIGURE 8. Intermediate business processes corresponding to Cus and Ven . (a) BP_{i1} , and (b) BP_{i2} .

Example 9: For the hidden cores C_{h1} and C_{h2} in Figure 7, we can generate their intermediate business processes using Algorithm 4, as shown in Figure 8.

At last, we present an approach to generate enforced business processes from intermediate business processes using coordination mapping. In essence, coordination mapping inserts coordination factors into intermediate business processes to ensure the fact that the generated enforced processes are correct and maximally permissive. In our context, coordination factors correspond to a class of special activities that are used to model coordination interactions to achieve coordination logic. It is worth noting that the communication between coordination factors is always synchronous.

Definition 10 (Coordination Mapping): Let $OP = BP_1 || \dots || BP_n$ be an original process, C be the core corresponding to OP , $BP = (S, A, T, s_0, F)$ be a business process in OP , $BP_i = (S_i, A_i, T_i, s_{i0}, F_i)$ be the intermediate business process corresponding to BP , and CT be the coordination transitions in C . Then, the enforced business process obtained by the coordination mapping of BP_i is an LTS $BP_e = (S_e, A_e, T_e, s_{e0}, F_e)$, where

- (1) $S_e = S_i \cup \{s_c | \forall t = (r, a, s) \in T_i \wedge a \notin A \wedge t \in CT\} \cup \{s_{c1}, s_{c2} | \forall t = (r, a, s) \in T_i \wedge a \in A \wedge t \in CT\}$;
- (2) $s_{e0} = s_{i0}$;
- (3) $F_e = F_i$;

$$\begin{aligned}
 (4) \quad A_e &= \{a | \forall t = (r, a, s) \in T_i \wedge a \in A \wedge t \notin CT\} \cup \\
 &\quad \{ \text{SYNC_1_}a, \text{SYNC_2_}a | \\
 &\quad \forall t = (r, a, s) \in T_i \wedge a \in A \wedge t \in CT\} \cup \\
 &\quad \{ \text{SYNC_1_}a, \text{SYNC_2_}a | \\
 &\quad \forall t = (r, a, s) \in T_i \wedge a \notin A \wedge t \in CT\}; \\
 (5) \quad T_e &= \{(r, \text{SYNC_1_}a, s_c), (s_c, \text{SYNC_2_}a, s) | \\
 &\quad \forall t = (r, a, s) \in T_i \wedge a \notin A \wedge t \in CT\} \cup \\
 &\quad \{(r, \text{SYNC_1_}a, s_{c1}), (s_{c1}, a, s_{c2}), \\
 &\quad (s_{c2}, \text{SYNC_2_}a, s) | \\
 &\quad \forall t = (r, a, s) \in T_i \wedge a \in A \wedge t \in CT\} \cup \\
 &\quad \{(r, a, s) | \forall t = (r, a, s) \in T_i \wedge a \in A \wedge t \notin CT\}.
 \end{aligned}$$

Based on Definition 10, we know that for a transition $t = (r, a, s) \in T_i$: 1) if $a \notin A$ and $t \in CT$, then two coordination factors SYNC_1_a and SYNC_2_a and a coordination state s_c are introduced, they are used to split the transition t into two new transitions $(r, \text{SYNC_1_}a, s_c)$ and $(s_c, \text{SYNC_2_}a, s)$, indicating that t is coordinated in state r through SYNC_1_a, and the coordination is completed in state s through SYNC_2_a; 2) if $a \in A$ and $t \in CT$, then two coordination factors SYNC_1_a and SYNC_2_a and two coordination states s_{c1} and s_{c2} are introduced, they are used to split the transition t into three new transitions $(r, \text{SYNC_1_}a, s_{c1})$, (s_{c1}, a, s_{c2}) and $(s_{c2}, \text{SYNC_2_}a, s)$, indicating that t is coordinated in state r through SYNC_1_a, and the coordination is completed in state s through SYNC_2_a; and 3) if $a \in A$ and $t \notin CT$, then t remains unchanged.

Example 10: Based on coordination mapping, we can generate two enforced business processes, as shown in Figure 9.

By composing enforced business processes, we can construct enforced processes.

Definition 11 (Enforced Process): Let $OP = BP_1 || \dots || BP_n$ be an original process, and BP_{e1}, \dots, BP_{en} be enforced business processes corresponding to BP_1, \dots, BP_n . Then, the enforced process corresponding to OP is $EP = BP_{e1} || \dots || BP_{en}$.

Example 11: By composing BP_{e1} and BP_{e2} in Figure 7, we can construct the enforced process $EP = BP_{e1} || BP_{e2}$, as shown in Figure 10.

By analyzing enforced processes built our approach, we can prove that they are effective. At first, we prove that they are correct.

To prove that enforced processes are correct, we present Lemma 2.

Lemma 2: Let $C = (S, A, T, s_0, F)$ be core, then for each state $s \in S$, there exists a final state $s_f \in F$, such that $s \xrightarrow{*} s_f$.

Proof: Let OP be the original process corresponding to C . According to Algorithm 2, we know that all invalid states and the transitions formed by these invalid states are removed from P_o while all valid states and the transitions formed these valid states are preserved in *core*. Since for each valid state $s \in S$, there exists a trace $\sigma = a_1 \dots a_n$, such that $s \xrightarrow{a_1} s_1 \dots s_{n-1} \xrightarrow{a_n} s_f$, where $s_f \in F$, we can derive that Lemma 1 holds. \square

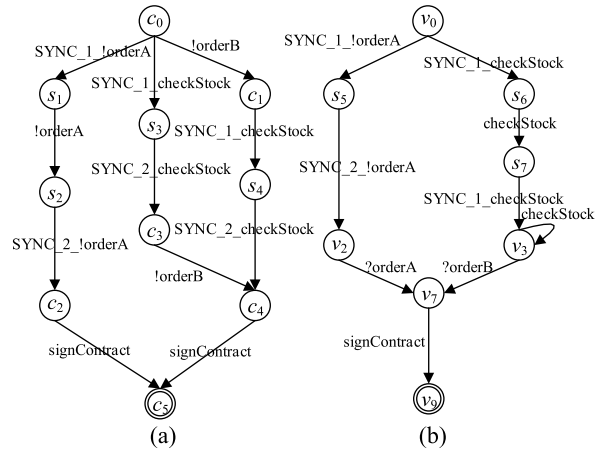


FIGURE 9. Enforced business processes corresponding to Cus and Ven. (a) BP_{e1} , and (b) BP_{e2} .

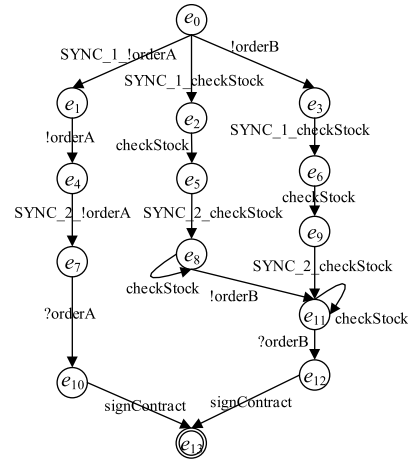


FIGURE 10. Enforced process EP.

Lemma 2 states that for each state in the core, it can always reach a final state.

Based on Lemma 2, we prove that enforced processes built by our approach are correct below.

Theorem 1: Let OP be an original process, and EP be the enforced process. Then, EP is correct.

Proof: The proof follows from the following observations. Support that $C = (S, A, T, s_0, F)$, $EP = (S_e, A_e, T_e, s_{e0}, F_e)$, and CT are the coordination transitions in C . For any state s_1 in S , we have

- (1) If $t = (s_1, a, s_2) \in T$ and $t \in CT$, then there exist three corresponding transitions $(s_{e1}, \text{SYNC_1_}a, s_{c1})$, (s_{c1}, a, s_{c2}) , $(s_{c2}, \text{SYNC_2_}a, s_{e2})$ in T_e ;
- (2) If $t = (s_1, a, s_2) \in T$ and $t \notin CT$, then there exists a corresponding transition (s_{e1}, a, s_{e2}) in T_e ;
- (3) There are no extra transitions generated from s_{e1} expect for corresponding transitions;
- (4) If $s_1 \in F$, then $s_{e1} \in F_e$;
- (5) If $s_{e1} \in F_e$, then $s_1 \in F$.

Based on Lemma 2, we know that for each state $s \in S$, there exists a final state $s_f \in F$, such that $s \xrightarrow{*} s_f$. Therefore, we

can derive that for each state $s_e \in S_e$, there exists a final state $s_{ef} \in F_e$, such that $s_e \xrightarrow{*} s_{ef}$.

According to Definition 6, we can conclude that P_e is correct. \square

Example 12: In Figure 10, we can see that all states in EP can reach the final state e_{13} . Therefore, we can derive that EP is correct.

Afterwards, we prove that they are maximally permissive.

To prove that enforced processes are maximally permissive, we present Lemma 3.

Lemma 3: Let C be a core, and EP be the enforced process generated from C , then EP preserves all legal traces in C without introducing additional traces in the case of neglecting the coordination factors in EP .

Proof: The proof follows from the following observations. Support that $C = (S, A, T, s_0, F)$, $EP = (S_e, A_e, T_e, s_{e0}, F_e)$, and CT are the coordination transitions in C . For any state s_1 in S , we have

- (1) If $t = (s_1, a, s_2) \in T$ and $t \in CT$, then there exist three corresponding transitions $(s_{e1}, SYNC_1_a, s_{c1})$, (s_{c1}, a, s_{c2}) , $(s_{c2}, SYNC_2_a, s_{e2})$ in T_e ;
- (2) If $t = (s_1, a, s_2) \in T$ and $t \notin CT$, then there exists a corresponding transition (s_{e1}, a, s_{e2}) in T_e ;
- (3) There are no extra transitions generated from s_{e1} expect for corresponding transitions;
- (4) If $s_1 \in F$, then $s_{e1} \in F_e$;
- (5) If $s_{e1} \in F_e$, then $s_1 \in F$.

Based on (1) - (5), we can derive that for each legal trace σ in C , there exists a unique corresponding legal trace σ' in EP , such that $\sigma = \sigma' \uparrow CF$, where CF refers to coordination factors in EP , and $\sigma \uparrow CF$ refers to a new trace generated by removing the coordination factors in CF from σ . Therefore, in case coordination factors are neglected, we can conclude that Lemma 3 holds. \square

Based on Lemma 3, we can prove that enforced processes preserve all legal traces in original processes without introducing additional traces in the case of neglecting coordination factors.

Theorem 2: Let OP be an original process, and EP be the enforced process. Then, EP is maximally permissive. That is, it preserves all legal traces in OP without introducing additional traces in the case of neglecting the coordination factors in EP .

Proof: Support that CF are the coordination factors in EP . According to Lemma 1, we know that C preserves all legal traces in OP without introducing additional traces. Therefore, we can derive that Theorem 2 holds based on Lemma 3. \square

Example 13: From C depicted in Figure 6 and EP shown in Figure 10, we can derive that there exists a mapping relationship between C and EP , as presented in Figure 11. That is, for each transition $t = (s_1, a, s_2)$ in C , two cases hold:

- (1) If t is a coordination transition (marked in red), then
 - 1) there exist three corresponding transitions $(s_{e1}, SYNC_1_a, s_{c1})$, (s_{c1}, a, s_{c2}) , $(s_{c2}, SYNC_2_a, s_{e2})$

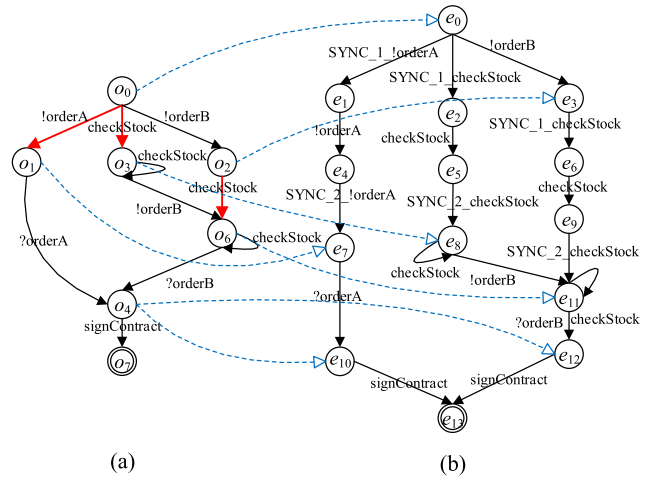


FIGURE 11. The mapping relationship between C and EP . (a) C , and (b) EP .

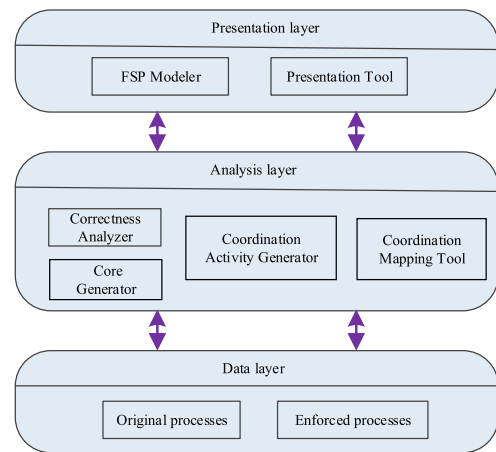


FIGURE 12. The architecture of cet.

in EP ; 2) if $s_1 \in \{o7\}$, then $s_{e1} \in \{e13\}$; 3) if $s_2 \in \{o7\}$, then $s_{e2} \in \{e13\}$; 4) if $s_{e1} \in \{e13\}$, then $s_1 \in \{o7\}$; and 5) if $s_{e2} \in \{e13\}$, then $s_2 \in \{o7\}$;

- (2) If t is not a coordination transition, then 1) there exist a corresponding transitions (s_{e1}, a, s_{e2}) in EP ; 2) if $s_1 \in \{o7\}$, then $s_{e1} \in \{e13\}$; 3) if $s_2 \in \{o7\}$, then $s_{e2} \in \{e13\}$; 4) if $s_{e1} \in \{e13\}$, then $s_1 \in \{o7\}$; and 5) if $s_{e2} \in \{e13\}$, then $s_2 \in \{o7\}$.

Based on Definition 2, we can conclude that EP is maximally permissive.

V. IMPLEMENTATION AND EXPERIMENTS

In this section, we first briefly describe the implementation of our approach. Then, we choose a set of real-world cases and conduct a series of experiments to validate our approach.

A. IMPLEMENTATION

Our approach is implemented as a prototype tool called cet. Figure 12 shows the architecture of cet. Based on separation of concerns, it involves three layers.

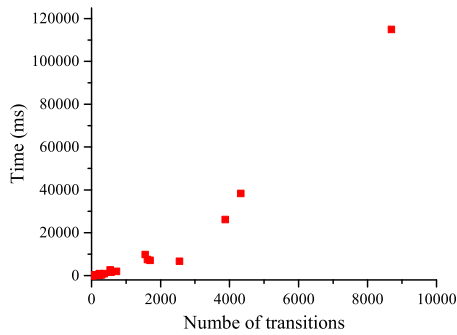


FIGURE 13. The relation between the model size and the time to enforce each case.

- (1) At the data layer, original processes are organized in FSP files, and enforced processes are specified in DOT language scripts.
- (2) The analysis layer involves four components: 1) the Correctness Analyzer can be used to analyze the correctness of original processes based on weak termination; 2) the Core Generator prunes the reachability graph of original processes to obtain their cores; 3) the Coordination Activity Generator generates a set of coordination activities from cores; and 4) the Coordination Mapping Tool can be used to generate enforced processes.
- (3) The presentation layer involves two components: 1) the FSP Modeler can be used to model original processes using FSP processes; and 2) the Presentation Tool can be used to display enforced processes using GraphViz (<http://www.graphviz.org/>).

In practice, business designers first specify an original process using FSP and input it into the FSP Translator. The FSP Translator then automatically converts the FSP specification of the original process into an LTS model. The Correctness Analyzer accepts the LTS model and then analyzes its correctness based on weak termination. In case the original process is partially correct, the Core Generator prunes the LTS model and generates its core. Based on the generated core, the Coordination Transition Generator generates a set of coordination transitions. Using these coordination transitions, the Coordination Mapping Tool produces a set of enforced business processes and an enforced process. The Presentation Tool leverages open-source graph visualization software GraphViz to display these enforced business processes and the enforced process.

B. EXPERIMENTS

In this section, we conduct a series of experiments to confirm the fact that whether our approach can improve the applicability compared to state-of-the-art approaches (i.e., the approaches in [10]–[13]) in the case of ensuring the effectiveness, and the scalability of our approach is not the focus of the current validation. The applicability and effectiveness of our approach are validated on real-world cases. All experiments

were carried out on a PC with 1.80GHz Processor and 16GB of RAM, running Windows 10.

Currently, there are no public collaborative business processes available for experiments [18]. Yet, in order to validate the applicability and effectiveness of our approach, we choose 30 real-world cases from existing research papers and the BPMN case base (<http://www.bpmn.org/>) as well as the process model matching case (<https://ai.wu.ac.at/emisa2015/contest.php>) for our experiments. These cases specify actual scenarios in different areas, which represent diverse and practical private processes.

Recall that our approach can enforce cases that are partially correct, and there is no need to enforce ones that are correct or incorrect [10]–[13]. Therefore, we manually modify the internal structure of each case to randomly inject errors (e.g., deadlocks, livelocks and unspecified receptions) before our experiments. In this way, we ensure that all the cases are partially correct.

For the thirty cases, we first analyze the applicability, and Table 2 presents the experimental results of some cases. In Table 2, $|N|$ and $|A|$ represent the number of participants and activities in the case, respectively. app , C_{enf} , and Mp are used to identify the applicability of the approach, the correctness of the enforced case and whether the enforced case is maximally permissive, respectively. Note that given a case, its corresponding enforced case can be generated by enforcing it using cet.

In Table 2, we can see that the approaches in [10]–[13] cannot be used to enforce these cases, i.e., app is “–” in Table 2. Concretely, since these cases cover behavioral abnormalities (e.g., deadlocks, livelocks and unspecified receptions) and these abnormalities in turn result in the fact that their reachability graph is not well-formed, the approach in [10] cannot be used to enforce these cases. For example, for $Ca-1$ (i.e., our motivating example Ord), the approach will not work as Ord 's reachability graph covers a livelock and this in turn results in it is not well-formed. For the approaches in [11]–[13], they also cannot be used to enforce these cases since synchronous communication or other behavioral abnormalities (e.g., livelocks and unspecified receptions) are introduced expect for deadlocks. For example, for $Ca-1$ (i.e., our motivating example Ord), since Ord covers both synchronous communication and livelocks, and this in turn results in the inability of the approaches in [11]–[13] to enforce these cases. In contrast, for all cases, our approach can complete correctness enforcement by inserting coordination factors into them, i.e., app is “+” in Table 2. Therefore, we can derive that the applicability of our approach can be greatly improved compared to the approaches in [10]–[13].

The effectiveness of our approach has been proved in Section 4. Here, we show the effectiveness of our approach through experiments, and Table 2 presents the experimental results of some cases. As expected, all enforced cases become correct and maximally permissive (i.e., both C_{enf} and Mp are “+”) once the additional coordination factors are added to cases. This confirms that our approach is effective.

TABLE 2. Experimental results of some cases.

Case	Size $ N / A $	The approaches in [10-13]		Our approach	
		app	C_{enf} Mp	app	C_{enf} Mp
Ca-01	2 / 8	-	+/- +/-	+	+ +
Ca-06	4 / 32	-	+/- +/-	+	+ +
Ca-10	6 / 39	-	+/- +/-	+	+ +
Ca-11	7 / 47	-	+/- +/-	+	+ +
Ca-12	8 / 55	-	+/- +/-	+	+ +
Ca-15	6 / 37	-	+/- +/-	+	+ +
Ca-19	6 / 41	-	+/- +/-	+	+ +
Ca-20	7 / 49	-	+/- +/-	+	+ +
Ca-21	8 / 57	-	+/- +/-	+	+ +
Ca-22	9 / 65	-	+/- +/-	+	+ +
Ca-29	2 / 51	-	+/- +/-	+	+ +
Ca-30	4 / 57	-	+/- +/-	+	+ +

In particular, since the approaches in [10]–[13] cannot be used to enforce these cases, their experimental results in terms of the effectiveness are not presented in Table 2, denoted by “+/-”.

Despite the fact that the scalability of our approach is not the focus of this paper. Yet, in order to identify the weaknesses of our approach to that it could be solved in future, during our experiments we record the time for enforcing every case. The minimum time is 2 ms, the maximum time is 114918 ms, the average time is 7387 ms, and the standard deviation is small, which can be ignored. Figure 13 depicts the relation between the model size and the time to enforce each case. Note that we identify the model size with the number of transitions as it is always larger than the number of states in each case.

In Figure 13, we can see that most cases have a smaller model size (i.e., the case contains few states, such as Ca-01), the time to achieve correctness enforcement is negligible. However, for medium-size cases, then more time is required yet the overall time for correctness enforcement is reasonable. For example, for Ca-22 with 8696 transitions, it takes about 1.9 min for cet to complete correctness enforcement. In practice, even it takes more time for cet to achieve correctness enforcement, this is not an issue since our enforcement is achieved at design time.

In theory, our approach needs to build the reachability graph of original processes when achieving correctness enforcement. Thus, it may suffer from the state-space explosion problem. How to solve the problem is not detailed in this paper, and our future work will alleviate the problem using various techniques such as stubborn set-based methods [30], unfolding methods [31], and BDD-based methods [32].

In particular, since the approaches in [10]–[13] cannot be used to enforce these cases, their experimental results in terms of the scalability are not presented in Figure 16.

VI. RELATED WORK

Our work is related to two research axes: correctness checking approaches; and (2) correctness enforcement approaches.

A. CORRECTNESS CHECKING APPROACHES

The existing work in this area can be divided into three categories, i.e., automata-based approaches, petri net-based approaches, and process algebra-based approaches.

1) AUTOMATA-BASED APPROACHES

Zhou *et al.* [19] proposed an automata-based approach for verifying mediated service interactions considering expected behavior. Their approach first employs Labeled Transition Systems (LTS) to model service protocols (each service protocol corresponds to a service-based process). Then, according to the adaptation mechanisms of a certain adapter, the approach generates the logic of the adapter that can be used to reconcile the mismatches between the protocols. Lastly, the reachability and liveness properties are verified using SPIN, and the result indicates whether the interaction is always adaptable. Flavio *et al.* [20] proposed a formal approach for modeling and verifying BPMN-based business process collaborations. Their approach first proposes the operational semantics for a relevant subset of BPMN elements that can be used to map BPMN-based collaborative business processes into Labeled Transition Systems, and then verifies correctness properties (e.g., the reachability and liveness properties) in terms of LTL formulae using the tool Maude.

2) PETRI NET-BASED APPROACHES

Aalst [4] presented an IOWF (Inter-Organizational Workflow) based approach for modeling and analyzing inter-organizational workflows. In this approach, the authors first use WF-net (Workflow net) [21] to model the business process of each party. Particularly, a WF-net is a special Petri net with exactly one source place i and exactly one sink place o . Additionally, if we add a transition e such that $e^\bullet = o$ and ${}^\bullet e = i$, then the WF-net is strongly connected. Then, they define two communication mechanisms, i.e., asynchronous communication and synchronous communication, to model interactions between business processes and compose the business process of each party in the collaboration using these two communication

mechanisms to obtain an inter-organizational workflow represented by IOWF. Lastly, they employ the unfolding operator to transform an IOWF into a WF-net, and verify the correctness of the IOWF in terms of the soundness property. Zhang *et al.* [22] presented an approach that relies on Petri nets and Pi calculus for modeling collaborative business processes. Their approach first employs Petri nets and Pi calculus to model the local processes in the collaboration and the interaction protocols between these local processes, respectively. Then, the approach defines the logic correctness of collaborative business processes based on soundness [4]. Lastly, a method verifying the logic correctness is presented, i.e., each local process is sound and the pi process modeling the interaction between local processes can be reduced into the process 0. Ge *et al.* [23] presented an approach that relies on Interaction-Oriented Petri Nets (IOPN) to model collaborative business processes. Their approach first uses IOPN to describe the workflow coordination between different organizations, i.e., collaborative business processes. Then, the approach introduces the notion of weak sound to define the logic correctness of collaborative business processes. At last, a decomposition approach with invariant analysis that can decompose a circuit-free and relaxed sound IOPN into a set of sequence diagrams is presented. This decomposition approach can avoid the state-space explosion problem, thereby improving the efficiency of correctness analysis. Yu *et al.* [8] presented a Petri nets-based approach for modeling and verifying cross-department processes considering different kinds of coordination patterns. Their approach extends WF-net by considering resource and message factors, namely RM_WF_Net, and additionally proposes several coordination patterns among different departments. By composing the business process of each party described by RM_WF_Net in the collaboration using these presented coordination patterns, a cross-department business process can be obtained and the soundness of the cross-department business process can be analyzed based on its reachability graph, i.e., 1) for any marking M that is reachable from the initial marking, the final marking can be reachable from M by executing a sequence of transitions, 2) if the final marking is reached, then there is exactly one token in the place o , and no tokens in the other places; and 3) there are no dead transitions in cross-department business process. To verify the correctness of complex business processes, Kheldoun *et al.* [24] proposed a formal verification approach based on high-level Petri nets. Their approach first uses Business Process Modeling Notation (BPMN) to model complex collaborative business processes. Then, the approach presents a formal semantics for BPMN using recursive ECATNets that can transform BPMN-based collaborative business processes into Petri nets. Finally, the correctness properties (e.g., the reachability property) with respect to collaborative business processes can be verified using the Maude LTL model checker. To verify the timed compatibility for mediation-aided web service composition, Du *et al.* [25] presented a three stages approach. First, stage 1 treats each service represented by the timed open workflow

net (ToN) in the composition as a fragment. Second, stage 2 transforms fragments into a time automata net (TAN) based on structure transformation and interactive message transformation. Finally, stage 3 checks all types of temporal constraints (i.e., related correctness properties) using UPPAAL. To avoid the verification of composite correctness.

3) PROCESS ALGEBRA-BASED APPROACHES

Wong and Gibbons [26] proposed an approach for modeling and verification of BPMN processes. Their approach introduces a semantic model for BPMN in the process algebra CSP (Communicating Sequential Process), and then specifies behavioral properties of BPMN diagrams and verifies the properties via automatic model checking tool FDR. To guarantee the success of Business Process Modelling (BPM), Mendoza *et al.* [27] proposed a conceptual framework for business processes compositional verification. Their approach transforms collaborative business processes specified by BPMN into Communicating Sequential Processes + Time (CSP+ T) processes, and then specifies the desired temporal properties in terms of Clocked Computation Tree Logic (CCTL) formulae and verifies the properties through Failure Divergence Refinement (FDR2). To improve the reliability for web service-based business process collaboration, Zhu *et al.* [28] presented an approach to model and verify web service-based business process collaboration based on model transformation. Their approach first establishes a modeling and verifying framework based on model transformation. Then, the approach presents a set of rules to transform BPE based private processes into CSP processes. Finally, the correctness of the composition of private processes are verified using the model checking tool Failure Divergence Refinement (FDR).

However, the above approaches focus on correctness checking and their main disadvantage is that if there are multiple errors in collaborative business processes, then multiple detections and adjustments are required. Since the diagnosis information after each detection is difficult to understand for non-experts and does not provide recipes on how to repair collaborative business processes, the repair process for them may be complicated. This will have a direct impact on the design of collaborative business processes.

B. CORRECTNESS ENFORCEMENT APPROACHES

Compared with correctness checking approaches, less attention is paid to the approach for correctness enforcement.

As the need for interservice compatibility analysis and indirect composition has gone beyond what the existing service composition/verification technologies can handle, Tan *et al.* [10] proposed an approach for compatibility enforcement based on Petri nets. Their approach first transforms a BPEL description (i.e., a web service-based business process) into a service workflow net, which is a kind of colored Petri net (CPN). Then, the approach analyzes the compatibility of two web service-based business processes, and then devise an approach to check whether there exists a

message mediator so that their composition does not violate the constraints imposed by either side. Finally, in case the message mediator exists, the approach generates it to assist the automatic composition of partially compatible business processes. However, the approach requires that the original process needs to satisfy a certain property when achieving correctness enforcement. That is, given two business processes N_1 and N_2 , and the data mapping I , there exists a mediator to glue N_1 and N_2 , if and only if $G(N_1, N_2, I)$ is well-formed, i.e., 1) for any marking M that is reachable from the initial marking, the final marking can be reachable from M by executing a sequence of transitions, 2) if the final marking is reached, then there is exactly one token in the place o , and no tokens in the other places except for message places. Additionally, unlike our approach, the approach deals with centralized orchestration-based business processes rather than fully decentralized choreography-based ones as mediators are introduced. To achieve correctness enforcement, Xiong *et al.* [11] proposed an approach for web service-based collaborative business processes based on Petri nets. Their approach first uses the composition net (i.e., the C-net) to model collaborative business processes. Then, the problem of behavioral compatibility among web services is hence transformed into the deadlock structure problem of a C-net. If there exist incompatibility cases, a policy based on appending additional information channels is proposed. Since information channels are introduced, the approach can deal with fully decentralized business processes. However, the approach requires that the original process needs to satisfy a certain property when achieving correctness enforcement. The property is strict, and it leads to the situation where the approach can only enforce original processes that has deadlocks. Additionally, the approach can't ensure that enforced processes are maximally permissive. Based on the approach in [11], Bi *et al.* [12], [13] proposed a novel compatibility enforcement approach based on Petri Nets. Their approach first employs service workflow nets to model service choreography (i.e., collaborative business processes). Afterwards, by combining structure and reachability analyses, the approach generates a controlled reduced graph for a collaborative business process, and then develops a maximally permissive state feedback control policy to prevent abnormality. Lastly, an optimal controller is constructed for the administrator of service composition to avoid deadlocks in service choreography. Like the approach in [11], the approach introduces information channels as well. Therefore, it can generate fully decentralized choreography-based enforced processes. Additionally, it ensures that enforced processes are maximally permissive. However, the approach requires that the original process needs to satisfy a certain property when achieving correctness enforcement. That is, there exists an optimal controller, if and only if the reduced reachability graph is well-formed, i.e., 1) for any marking M that is reachable from the initial marking, the final marking can be reachable from M by executing a sequence of transitions, 2) if the final marking is reached, then there is exactly one token

in the place o , and no tokens in the other places. Like the approach in [11], the approach can only address the situation where there are deadlocks in original processes.

C. SUMMARY OF EXISTING WORK

Based on this literature review, we can see that existing approaches (e.g., [4], [8], [19]–[28]) mainly focus on correctness checking. Since iterative detection and adjustment are required during achieving correctness, this will greatly affect the modeling efficiency of collaborative business processes. Additionally, as diagnosis information after each detection is difficult to understand for non-experts and does not provide recipes on how to repair collaborative business processes, the repair process for them may be complicated. This will further weaken the modeling efficiency of collaborative business processes. Several approaches (e.g., [10]–[13]) concentrate on correctness enforcement, but run into the applicability problem as they fail to consider both synchronous and asynchronous communication and need to put additional restrictions on original processes.

In comparison, since our approach considers both synchronous and asynchronous communication and achieves correctness enforcement by inserting coordination factors into original processes, it can resolve the applicability problem in existing approaches (e.g., [10]–[13]) in the case of ensuring the effectiveness.

VII. CONCLUSION

The correctness analysis of collaborative business processes is considered to be an important issue in Business Process Management (BPM). In this paper, we propose a novel correctness enforcement approach for collaborative business processes. Since enforced processes built by our approach are correct and maximally permissive, repeated detection and adjustment in existing correctness checking approaches are avoided. Additionally, as our approach considers both synchronous and asynchronous communication and does not need to put any restrictions on original processes, it is more applicable than existing correctness enforcement approaches.

The future work will be mainly carried out in the following three aspects: 1) Since our approach needs to build the reachability graph of original processes when achieving correctness enforcement, it may suffer from the state-space explosion problem. How to achieve correctness enforcement more effectively will be discussed in our future work; 2) In this paper, we employ LTSs to specify collaborative business processes. Compared with other graphical specifications such as BPMN, UML and Petri nets, the model described by LTSs is complicated and difficult to understand for business designers, and this in turn limits the acceptance of our approach in practice. The future work will improve the readability of enforced processes; and 3) In essence, weak termination can be seen as a fundamental correctness criterion yet actual requirements may be diverse such as temporal constraints between activities. Our future work will consider diverse collaborative requirements when achieving correctness enforcement.

REFERENCES

- [1] T. Jin, J. Wang, Y. Yang, L. Wen, and K. Li, "Refactor business process models with maximized parallelism," *IEEE Trans. Services Comput.*, vol. 9, no. 3, pp. 456–468, May/June 2016.
- [2] A. Yousfi, A. de Freitas, A. K. Dey, and R. Saidi, "The use of ubiquitous computing for business process improvement," *IEEE Trans. Services Comput.*, vol. 9, no. 4, pp. 621–632, Jul./Aug. 2016.
- [3] W. Song, F. Chen, H. Jacobsen, X. Xia, C. Ye, and X. Ma, "Scientific workflow mining in clouds," *IEEE Trans. Parallel Distrib. Syst.*, vol. 28, no. 10, pp. 2979–2992, Oct. 2017.
- [4] M. P. Van Der Aalst, "Modeling and analyzing interorganizational workflows," in *Proc. Int. Conf. Appl. Concurrency Syst. Design*. Los Alamitos, CA, USA, Mar. 1998, pp. 262–272.
- [5] W. Song and H.-A. Jacobsen, "Static and dynamic process change," *IEEE Trans. Services Comput.*, vol. 11, no. 1, pp. 215–231, Jan./Feb. 2018.
- [6] Y. Li, Z. Luo, J. Yin, L. Xu, Y. Yin, and Z. Wu, "Enterprise Pattern: Integrating the business process into a unified enterprise model of modern service company," *Enterprise Inf. Syst.*, vol. 11, no. 1, pp. 37–57, Jan. 2017.
- [7] W. Yu, C. G. Yan, Z. Ding, C. Jiang, and M. Zhou, "Modeling and verification of online shopping business processes by considering malicious behavior patterns," *IEEE Trans. Autom. Sci. Eng.*, vol. 13, no. 2, pp. 647–662, 2016.
- [8] W. Yu, C. Yan, Z. Ding, C. Jiang, and M. Zhou, "Analyzing E-commerce business process nets via incidence matrix and reduction," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 1, pp. 130–141, Jan. 2018.
- [9] Q. Zeng, F. Lu, C. Liu, H. Duan, and C. Zhou, "Modeling and verification for cross-department collaborative business processes using extended Petri nets," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 45, no. 2, pp. 349–362, Feb. 2015.
- [10] W. Tan, Y. Fan, M. Zhou, and M. Zhou, "A Petri net-based method for compatibility analysis and composition of Web services in business process execution language," *IEEE Trans. Autom. Sci. Eng.*, vol. 6, no. 1, pp. 94–106, Jan. 2009.
- [11] P. Xiong, Y. Fan, and M. Zhou, "A Petri net approach to analysis and composition of Web services," *IEEE Trans. Syst., Man, Cybern. A, Syst. Humans*, vol. 40, no. 2, pp. 376–387, Mar. 2010.
- [12] J. Bi, H. Yuan, and M. Zhou, "A Petri net method for compatibility enforcement to support service choreography," *IEEE Access*, vol. 4, pp. 8581–8592, 2017.
- [13] J. Bi, H. Yuan, and W. Tan, "Deadlock prevention for service orchestration via controlled Petri nets," *J. Parallel Distrib. Comput.*, vol. 124, pp. 92–105, Feb. 2019.
- [14] J. E. Hopcroft and J. D. Ullman, *Introduction to Automata Theory, Languages, and Computation*. Boston, MA, USA: Addison-Wesley, 1979.
- [15] N. Lohmann, "Compliance by design for artifact-centric business processes," *Inf. Syst.*, vol. 38, no. 4, pp. 606–618, Jun. 2013.
- [16] S. Blom and S. Orzan, "Distributed branching bisimulation reduction of state spaces," *Electron. Notes Theor. Comput. Sci.*, vol. 89, no. 1, pp. 99–113, Sep. 2003.
- [17] D. Ciolek, V. Braberman, and N. Dippolito, "Interaction models and automated control under partial observable environments," *IEEE Trans. Softw. Eng.*, vol. 43, no. 1, pp. 19–33, Jan. 2017.
- [18] M. Borkowski, W. Fdhila, M. Nardelli, S. Rinderle-Ma, and S. Schulte, "Event-based failure prediction in distributed business processes," *Inf. Syst.*, vol. 81, pp. 220–235, Mar. 2018. doi: 10.1016/j.is.2017.12.005.
- [19] Z. Zhou, L. T. Yang, S. Bhiri, L. Shu, N. Xiong, and M. Hauswirth, "Verifying mediated service interactions considering expected behaviours," *J. Netw. Comput. Appl.*, vol. 34, no. 4, pp. 1043–1053, Jul. 2011.
- [20] F. Corradini, F. Fornari, A. Polini, B. Re, and F. Tiezzi, "A formal approach to modeling and verification of business process collaborations," *Sci. Comput. Program.*, vol. 166, pp. 35–70, Nov. 2018.
- [21] M. P. Van der Aalst, "The application of Petri nets to workflow management," *J. Circuits Syst. Comput.*, vol. 8, no. 1, pp. 21–66, Feb. 1998.
- [22] L. Zhang, Y. Lu, and F. Xu, "Unified modelling and analysis of collaboration business process based on Petri nets and Pi calculus," *IET Softw.*, vol. 4, no. 5, pp. 303–317, Oct. 2010.
- [23] J. Ge and H. Hu, "A decomposition approach with invariant analysis for workflow coordination," *Chin. J. Comput.*, vol. 35, no. 10, pp. 2169–2181, Oct. 2012.
- [24] A. Kheldoun, K. Barkaoui, and M. Ioualalen, "Formal verification of complex business processes based on high-level Petri nets," *Inf. Sci.*, vols. 385–386, pp. 39–54, Apr. 2017.
- [25] Y. Du, B. Yang, and H. Hu, "Model checking of timed compatibility for mediation-aided Web service composition: A three stage approach," *Expert Syst. Appl.*, vol. 112, no. 1, pp. 190–207, Dec. 2018.
- [26] P. Y. H. Wong and J. Gibbons, "Formalisms and applications of BPMN," *Sci. Comput. Program.*, vol. 76, no. 8, pp. 633–650, Aug. 2011.
- [27] L. E. Mendoza, M. I. Capel, and M. A. Pérez, "Conceptual framework for business processes compositional verification," *Inf. Softw. Technol.*, vol. 54, no. 2, pp. 149–161, Feb. 2012.
- [28] Y. Zhu, Z. Huang, and H. Zhou, "Modeling and verification of Web services composition based on model transformation," *Softw., Pract. Exper.*, vol. 47, no. 5, pp. 709–730, May 2017.
- [29] K. Ajay, P. Pascal, and S. Gwen, "Checking business process evolution," *Sci. Comput. Program.*, vol. 170, no. 15, pp. 1–26, Jan. 2019.
- [30] F. M. Bønneland, J. Dyhr, P. G. Jensen, M. Johannsen, and J. Srba, "Stubborn versus structural reductions for Petri nets," *J. Log. Algebr. Methods Program.*, vol. 102, pp. 46–63, Jan. 2019.
- [31] D. Xiang, G. Liu, and C. Yan, "Detecting data inconsistency based on the unfolding technique of Petri nets," *IEEE Trans. Ind. Informat.*, vol. 13, no. 6, pp. 2995–3005, Dec. 2017.
- [32] C. Latsou, S. J. Dunnett, and L. M. Jackson, "A new methodology for automated Petri net generation: Method application," *Rel. Eng. Syst. Saf.*, vol. 185, pp. 113–123, Dec. 2019.



QI MO received the B.S. degree from Huaibei Normal University, Huaibei, China, in 2009, the M.S. degree from Yunnan University, Kunming, China, in 2012, and the Ph.D. degree in software engineering from Yunnan University, Kunming, China, in 2015. He is currently a Lecturer with Yunnan University, Kunming, China. He has led or participated in many projects supported by the National Natural Science Foundation and key projects at provincial levels. He has authored over 10 papers in journals and conference proceedings. His research interests include formal methods and business process management (BPM).



LIRUI BAI received the B.S. degree from the Tianjin University Renai College, Tianjin, China, in 2010, and the M.S. degree from Yunnan University, Kunming, China, in 2012. She is currently a Lecturer with the Tianjin University Renai College. She has participated in some projects supported by the National Natural Science Foundation and key projects at provincial levels. She has authored over five papers in journals and conference proceedings. Her research interests include formal methods, software engineering, and big data.



FEI DAI received the B.S. and M.S. degrees and the Ph.D. degree in software engineering from Yunnan University, Kunming, China, in 2005, 2008, and 2011, respectively. He is currently a Professor with Southwest Forestry University, Kunming. He has led or participated in many projects supported by the National Natural Science Foundation and key projects at provincial levels. He has authored over 50 papers in journals and conference proceedings. His research interests include business process management (BPM) and software engineering.



JIANGLONG QIN received the B.S. and M.S. degrees and the Ph.D. degree in software engineering from Yunnan University, Kunming, China, in 2006, 2009, and 2018, respectively, where he is currently a Lecturer. He has participated in many projects supported by the National Natural Science Foundation and key projects at provincial levels. He has authored over eight papers in journals and conference proceedings. His research interests include formal methods and business process management (BPM).



ceedings. His research interests include formal methods and software engineering.

ZHONGWEN XIE received the B.S. degree from the Harbin Institute of Technology, Harbin, China, in 2009, and the M.S. degree and the Ph.D. degree in software engineering from Yunnan University, Kunming, China, in 2009 and 2012, respectively, where he is currently a Lecturer. He has participated in many projects supported by the National Natural Science Foundation and key projects at provincial levels. He has authored over nine papers in journals and conference proceedings.



research interests include software process and software engineering.

TONG LI received the B.S. and M.S. degrees from Yunnan University, Kunming, China, in 1983 and 1988, respectively, and the Ph.D. degree in software engineering from De Montfort University, Leicester, U.K., in 2007. He is currently a Professor with Yunnan University. He has led or participated in many projects supported by the National Natural Science Foundation and key projects at provincial levels. He has authored over 100 papers in journals and conference proceedings.

• • •