Received December 6, 2018, accepted December 16, 2018, date of publication December 25, 2018,
date of current version March 25, 2019.

Digital Object Identifier 10.1109/ACCESS.2018.2889715

Development of Private Processes: A Refinement Approach

QI MO[1,2], FEI DAI[2,3], DI LIU[1], JIANGLONG QIN[1,2], ZHONGWEN XIE[1,2], AND TONG LI[1,2]
[1]School of Software, Yunnan University, Kunming 650091, China
[2]Key Laboratory of Software Engineering of Yunnan Province, Kunming 650091, China
[3]School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming 650091, China

Corresponding author: Fei Dai (59671019@qq.com)

This work was supported in part by the Project of National Natural Science Foundation of China under Grant 61862065, Grant 61462095, Grant 61702442, Grant 61662085, and Grant 61379032, in part by the Application Basic Research Project in Yunnan Province under Grant 2016FB102 and Grant 2018FB105, in part by the Open Foundation of Key Laboratory for Software Engineering of Yunnan Province under Grant 2017SE201 and Grant 2016SE202, in part by the Yunnan Province Young Academic and Technical Leaders Funds for Training under Grant C6143002, and in part by the Project of Yunnan Provincial Department of Education Science Research Fund under Grant 2017ZZX227.

ABSTRACT Private processes are the basis to construct the collaborative business processes, and their correctness has a direct impact on the correctness of collaborative business processes. Thus, the modeling and correctness of private processes is a key issue that business designers should consider at design time. To this end, we propose an approach for constructing correct private processes in a refinement manner. In this approach, we first present a formal model for private processes and abstract the control flow of the model into four basic blocks. Then, we derive a set of refinement rules for the four basic blocks and present an approach that relies on the refinement rules for constructing correct private processes. Finally, we prove that the private process established by our approach is correct at the syntax and semantic levels, and thus a subsequent correctness verification is avoided. Our approach is validated through a case study, and the results show that the approach is more effective than the existing work in terms of modeling private processes.

INDEX TERMS Collaborative business process, private process, basic block, refinement rule, correctness.

I. INTRODUCTION

With the rapid development of economic globalization, the business model of enterprises in the new information era has undergone major changes [1]. That is, the business model has evolved from individual enterprises with an independent development pattern into multiple enterprises with a cooperation pattern. This means that no enterprise is isolated in the modern business environment [2]–[3]. To achieve a common business goal, business processes in the collaboration need to cross organizational boundaries and then to interact with each other to form a relatively stable process view. Such a process view is widely known as a collaborative business process [24], [42].

Collaborative business processes that are regarded as a crucial enabling technology can achieve the integration of cross-organizational enterprise applications. Currently, such an enabling technology has been widely used in multiple industries. Some prominent examples are enterprise information systems based on PAIS (Process-Aware Information Systems) [4], e-commerce business processes [5], and medical business processes [6]. In these systems, to ensure the correct implementation of collaborative business processes, various modeling approaches are presented. Generally, these approaches can be classified into two kinds: the top-down approach [9], [10] and the bottom-up approach [6]–[7]. The top-down approach requires business designers to give a global contract and transformation rules and then to generate the business process for each organization from the contract using these given transformation rules. Nevertheless, the top-down approach that is usually restricted by transformation rules only establishes business processes with limited control structures (i.e., sequence, switch, flow, and loop control structures), which is not flexible and applicable in practice. In contrast, the bottom-up approach allows each organization to define its own private process independently and then construct a collaborative business process by composing these private processes together. Therefore, the bottom-up approach provides participating organizations freedom to develop their own business processes independently, and this in turn solves the inflexible and inapplicable issues of the top-down approach. Hence, in this paper we focus on the bottom-up approach. Particularly, a private process is owned by one organization and describes the internal business logic as well as the message exchanges this organization is engaged [11], [18]. In practice, private processes correspond to the executable processes of organizations and are usually described using the Business Process Execution Language (BPEL) [18].

However, in the bottom-up modeling approach, private processes in the collaboration are developed by different organizations, and cannot foresee all potential interactions at design time. Thus, in actual collaboration, the execution of the established model may produce undesirable outcomes such as deadlocks and livelocks. Thus, the correctness analysis of collaborative business processes is considered to be an important issue in Business Process Management (BPM) [7].

At present, multiple approaches for the correctness analysis of collaborative business processes have presented. However, these approaches mainly focus on *composite correctness*, but fail to consider *self-correctness*. Composite correctness means that the composition of private processes in the collaboration can terminate correctly, while self-correctness means that a private process under a perfect environment is considered to be correct at both the syntax and semantic levels [7]. Note that given a private process, its external environment in our context refers to the other private processes in the collaboration, and the term *perfect* implies that whenever the private process is expected to receive or send a message, its environment always sends or receives such the message. For a private process, if it does not conform to self-correctness, then composite correctness may be violated [7]. Additionally, in the correctness analysis of collaborative business processes, self-correctness should be met before composite correctness [7]. Hence in this paper we focus on self-correctness.

Self-correctness is an important aspect that business designers should consider at design time. Neglecting the aspect will make the construction of private processes lack a guideline, and this in turn affects modelling efficiency in practice. Additionally, self-correctness is usually analyzed based on the state space of private processes. If the structure of private processes is complex such as the number of parallel branches in private processes is large or the length of these parallel branches is longer, then the state-space explosion problem may occur. This problem will directly affect the correctness analysis of private processes, and eventually affect the correctness analysis of collaborative business processes [7].

To address these problems, we propose an approach that relies on four basic blocks (i.e., sequence, selection, concurrency and iteration blocks) for constructing correct private processes. This approach allows participating organizations to construct their private processes in a refinement manner, and ensures that the established private processes are correct at both the syntax and semantic levels. As a subsequent correctness analysis for private processes can be avoided, the efficiency of the correctness analysis of collaborative business processes can be improved eventually [7]. Note that we focus in this paper on structured private processes since they are close to BPEL, which is the standard language for implementing private processes using service-oriented technologies. Additionally, structured private processes allow for a simple definition of refinement rules [41]. Particularly, for unstructured processes, several approaches have been presented to convert an unstructured process into a structured one or a BPEL process while preserving its behavior [22], [23].

The main contributions of this paper are as follows:

(1) We propose a method that can be used to abstracting the control flow of private processes into four basic blocks. Based on the four basic blocks, we derive a set of refinement rules and then present a method to build private processes through stepwise refinement.
(2) We prove that the private processes built using our approach are correct at the syntax and semantic levels.
(3) The case study shows that our method is easy to use for business designers and can avoid the state-space explosion problem in existing approaches.

The remainder of this paper is organized as follows. Section 2 discusses related work. Section 3 presents the formal definition for private processes. Section 4 presents the method for constructing private processes and proves the established private processes are correct. Section 5 evaluates the effectiveness and efficiency of the approach using a case study. Section 6 concludes this paper.

II. MOTIVATION EXAMPLE

In this section, we present a simplified cell phone design process (*CPD*) used throughout the paper to illustrate our approach. *CPD* involves 2 participants, i.e., a hardware designer (denoted as *HarDesigner*) and a software developer (denoted as *SofDeveloper*). The interaction between the two participants is depicted in Figure 1, which includes the following steps:

(1) To design a new cell phone, *HarDesigner* first concurrently designs its motherboard and peripherals according to actual needs, and sends the motherboard parameters *motPar* to *SofDeveloper*;
(2) After receiving *motPar*, *SofDeveloper* develops the software *motSoft* for the motherboard;
(3) After *motSoft* is developed, *SofDeveloper* sends *motSoft* to *HarDesigner*;
(4) After receiving *motSoft*, *HarDesigner* integrates both the software *motSoft* and the hardware (i.e., the motherboard and peripherals) to complete the design of the phone.

In Figure 1, we can see that the correctness of *CPD* involves two aspects: self-correctness and composite correctness. Self-correctness means that the private processes of both *HarDesigner* and *SofDeveloper* are correct. Composite correctness means that the composition of their private processes is correct.

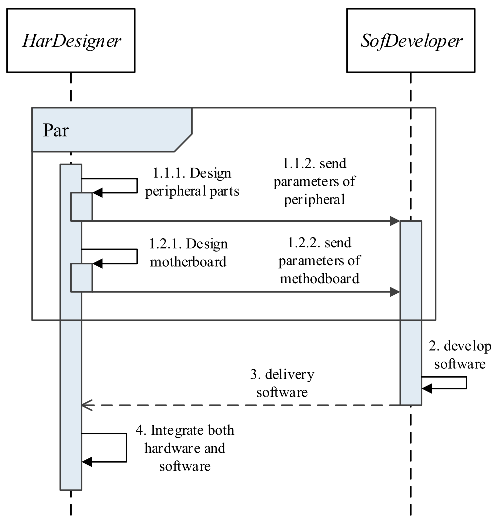
HarDesigner
SofDeveloper
Par
1.1.1. Design peripheral parts
1.1.2. send parameters of peripheral
1.2.1. Design motherboard
1.2.2. send parameters of methodboard
2. develop software
3. delivery software
4. Integrate both hardware and software

FIGURE 1. The interaction of *CPD* depicted by UML sequence diagram.

However, the existing work mainly concentrates on composition correctness yet ignores self-correctness. In fact, during the execution of *CPD*, self-correctness has a direct impact on composite correctness. For example, if the private process of *HarDesigner* has a deadlock during execution, then composite correctness must be violated. Additionally, if the structure of the private processes of *HarDesigner* and *SofDeveloper* is complex, then the state-space explosion problem may occur, and this in turn results in low efficiency when analyzing composite correctness. Therefore, how to construct correct private processes to avoid subsequent correctness verification for private processes remains largely open.

III. PRELIMINARIES

In this section, we present a formal model for private processes. A private process consists of tasks and the control flow between tasks. A task is an atomic unit of work or activity in a business process. Formally, a task is defined as follows.

Definition 1 (Task): A task is a 4-tuple *Tasks* = (*name*, *org*, *MsgRec*, *MsgSend*), where

1) *name* is the name of *task*;
2) *org* is the organization to which *task* belongs;
3) *MsgRec* denotes a set of messages which is received by task before it is executed;
4) *MsgSend* denotes a set of messages which is sent by *task* after it is executed.

Given a task *ta*, we use *ta.name*, *ta.org*, *ta.MsgRec* and *ta.MsgSend* to refer to different components of *ta*. In particular, if the condition $ta.MsgRec \neq \emptyset \vee ta.MsgSend \neq \emptyset$ holds, then we refer to *ta* as a communication task, otherwise *ta* is a local task. A communication task makes its private process interact with its external environment, as the task sends messages to its external environment or receives messages from its external environment.

The control flow specifies sequential constraints between a set of tasks using control structures such as sequence, switch and flow structures. In our work, we employ Petri nets to model the control flow, as they have an intuitive graphical representation, strict formal semantics and are suitable for expressing various control structures.

Definition 2 (Control Flow): The control flow is a basic Petri net $Flow = (P, T; F, M_0, i, o)$, where

(1) P is a finite set of places;
(2) T is a finite set of transitions;
(3) $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation;
(4) M_0 is the initial marking, such that for any $p \in P, p \neq i$: $M_0(p) = 0$ and $M_0(i) = 1$;
(5) i and o are two special places, which represent source and sink places, respectively.

With the concepts of task and control flow, a private process is formally defined as follows.

Definition 3 (Business Process): A private process is a 3-tuple $BP = (Tasks, Flow, f)$, where

(1) *Tasks* is a set of tasks. For any $task \in Tasks$, *task* conforms to Definition 1;
(2) *Flow* is the control flow;
(3) $f: T \rightarrow Tasks$ is a function that assigns tasks to transitions.

In essence, the control flow of a private process describes the internal business logic of the private process. Additionally, for a task *task* in the private process, if *task* is a communication task, meaning that the private process needs to interact with its external environment when executing *task*. By defining that the messages need to be received before the execution of *task* and the messages need to be sent after the execution of *task* (see Definition 1), we can well specify the interaction between the private process and its external environment.

For a private process *BP* and its external environment *E*, the running state of *BP* can be represented with a set of markings $M \subseteq Flow.P$. In practice, the execution of the tasks in *BP* is not only restricted by the control flow of *BP*, but may also be affected by *E*. A task is enabled under the marking M, denoted as $M[t>$, iff its corresponding transition t is enabled under M, and the messages that *task* needs to receive is available, then a new marking M' can be reached by the execution of *task*, denoted by $M[task > M'$.

By Definition 2, we can derive that the control flow of private processes is a workflow net [13]. Therefore, we define the correctness of the control flow of private processes using soundness [13].

Definition 4 (Sound): A business process *BP* is sound, iff

(1) $\forall M: M_0[\sigma_1 > M \rightarrow M[\sigma_2 > M_f$, where M_f is the final marking of *BP*, such that for any $p \in BP.Flow.P, p \neq o$: $M_f(p) = 0$ and $M_f(o) = 1$;
(2) $\forall M': M_0[\sigma > M' \wedge M' \geq o \rightarrow M' = o$;
(3) $\forall t \in BP.Flow.T$, there exist two markings M, M' such that $M_0[\sigma > M[t > M'$.

where $\sigma, \sigma_1, \sigma_2 \in BP.Flow.T*$; $M[\sigma > M'$ indicates that *BP* reaches a new marking M' from the marking M by executing a sequence of transitions σ.

IV. BASIC BLOCKS AND REFINEMENT RULES

In this section, we first define four basic blocks. Then, we derive the refinement rules for the four basic blocks. Finally, we present an approach for constructing private processes, and analyze the correctness of the established private processes.

A. BASIC BLOCKS

As the introduction illustrates, we focus in this paper on structured private processes such as BPEL processes. Typically, structured processes consist of four basic control flow structures, i.e., sequence, selection, iteration and concurrency structures [41]. In essence, these four basic control flow structures correspond to the sequence, switch, flow, and loop control structures in BPEL.

Theoretically, we can't produce a private process with arbitrary structure by refining these four basic blocks [25]. However, according to our investigation, most of private processes independently developed in reality (such as BPEL-based service processes) are structured. Hence, our approach that constructs the correct private process by refining the basic blocks is applied well in practice.

In the following, we formally define four basic blocks for the above four control structures. Note that in our work, we define private processes (see Definition 3) that realize the separation of the control flow and tasks. Thus, we can abstract the control flow of private processes into four types of basic blocks (i.e., sequence, selection, concurrency, and iterative blocks) without involving tasks, and each basic block corresponds to a control structure. For the sake of simplicity, Definitions 4-11 and Rules 1-4 only consider the control flow, and no tasks are involved.

A basic block is a structural fragment with exactly one entry and exactly one exit in a private process. In our context, we employ Petri nets to model basic blocks. Formally, a basic block is defined as follows.

Definition 5 (Basic Block): A basic block is a 7- tuple $B = (P, T; F, A_e, A_x, R_e, R_u)$, where

(1) P is a set of places;
(2) T is a set of transitions;
(3) $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation;
(4) $A_e, A_x \subseteq T$ are the entry and exit of the basic block, respectively;
(5) $R_e, R_u \subseteq T$ denote a set of refinable transitions and a set of non-refinable transitions, respectively. Particularly, refinable transitions are transitions that can be replaced with basic blocks when constructing private processes, while non-refinable transitions have only been added for routing purposes, i.e., preserving the integrity of basic blocks.

In Figure 2 (a), a sequence block depicts the sequential execution of the tasks corresponding to the transitions t_i and t_j.

Definition 6 (Sequence Block): Let $B = (P, T; F, A_e, A_x, R_e, R_u)$ be a basic block, we call B a sequence block, iff B satisfies the following conditions:

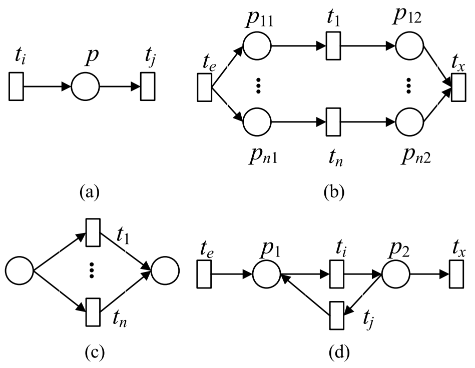

FIGURE 2. The basic blocks.

(1) $P = \{p\}$;
(2) $T = \{t_i, t_j\}$;
(3) $F = \{(t_i, p), (p, t_j)\}$;
(4) $A_e = \{t_i\}$;
(5) $A_x = \{t_j\}$;
(6) $R_e = \{t_i, t_j\}$;
(7) $R_u = \emptyset$.

In Figure 2 (b), a concurrency block depicts the concurrent execution of the tasks corresponding to the transitions $t_1, \ldots, t_n$.

Definition 7 (Concurrency Block): Let $B = (P, T; F, A_e, A_x, R_e, R_u)$ be a basic block, we call B a concurrency block, iff B satisfies the following conditions:

(1) $P = \{p_{11}, p_{12}, \ldots, p_{n1}, p_{n2}\}$;
(2) $T = \{t_1, \ldots, t_n, t_e, t_x\}$;
(3) $F = \{(t_e, p_{11}), \ldots, (t_e, p_{n1}), (p_{11}, t_1), \ldots, (p_{n1}, t_n), (t_1, p_{12}), \ldots, (t_n, p_{n2}), (p_{12}, t_x), \ldots, (p_{n2}, t_x)\}$;
(4) $A_e = \{t_e\}$;
(5) $A_x = \{t_x\}$;
(6) $R_e = \{t_1, \ldots, t_n\}$;
(7) $R_u = \{t_e, t_x\}$.

In Figure 2 (c), a selection block depicts the selective execution of the tasks corresponding to the transitions $t_1, \ldots, t_n$.

Definition 8 (Selection Block): Let $B = (P, T; F, A_e, A_x, R_e, R_u)$ be a basic block, we call B a selection block, iff B satisfies the following conditions:

(1) $P = \emptyset$;
(2) $T = \{t_1, \ldots, t_n\}$;
(3) $F = \emptyset$;
(4) $A_e = \{t_1, \ldots, t_n\}$;
(5) $A_x = \{t_1, \ldots, t_n\}$;
(6) $R_e = \{t_1, \ldots, t_n\}$;
(7) $R_u = \{t_e, t_x\}$.

In Figure 2 (d), an iteration block depicts the iterative execution of the tasks corresponding to the transitions t_i, t_j.

Definition 9 (Iteration Block): Let $B = (P, T; F, A_e, A_x, R_e, R_u)$ be a basic block, we call B an iteration block, iff B satisfies the following conditions:

(1) $P = \{p_1, p_2\}$;
(2) $T = \{t_i, t_j, t_e, t_x\}$;

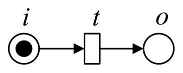

FIGURE 3. The source control flow of *HarDesigner*.

(3) $F = \{(t_e, p_1), (p_1, t_i), (t_i, p_2), (p_2, t_j), (t_j, p_1), (p_2, t_x)\}$;
(4) $A_e = \{t_e\}$;
(5) $A_x = \{t_x\}$;
(6) $R_e = \{t_i, t_j\}$;
(7) $R_u = \{t_e, t_x\}$.

B. REFINING BASIC BLOCKS

Based on the above four basic blocks, we can derive refinement rules. At first, we introduced the notion of source control flow. A source control flow saves as the start point for constructing the control flow of a private process. Formally, we define a source control flow as follows.

Definition 10 (Source Control Flow): A source control flow is a 6-tuple $SP = (P, T; F, R_e, R_u, M_0)$, where
(1) $P = \{i, o\}$;
(2) $T = \{t\}$;
(3) $F \subseteq \{(i, t), (t, o)\}$;
(4) $R_e = \{t\}$;
(5) $R_u = \emptyset$;
(6) M_0 is the initial marking, such that for any $p \in P, p \neq i$: $M_0(p) = 0$ and $M_0(i) = 1$;
(7) $i, o \in P$ are two special places, which represent source and sink places, respectively.

Afterwards, we can generate a refined control flow by refining transition t in a source control flow. Formally, a refined control flow is defined as follows.

Example 1: To develop the private process of *HarDesigner* in *CPD*, we need to first define its source control flow, as shown in Figure 3.

Definition 11 (Refined Control Flow): A refined control flow is a 6-tuple $RP = (P, T; F, R_e, R_u, M_0)$, where
(1) P is a finite set of places;
(2) T is a finite set of transitions;
(3) $F \subseteq (P \times T) \cup (T \times P)$ is a flow relation;
(4) R_e is a set of refinable transitions;
(5) R_u is a set of non-refinable transitions;
(6) M_0 is the initial marking, such that for any $p \in P, p \neq i$: $M_0(p) = 0$ and $M_0(i) = 1$;
(7) $i, o \in P$ are two special places, which represent source and sink places, respectively.

In essence, source control flows can be regarded as initial refined control flows. By iteratively refining refined control flows, we are able to generate the control flow of private processes.

With the concept of refined control flow, we present several refinement rules based on four basic blocks as follows.

The refinement rule based on sequence blocks is depicted in Figure 4(a).

Rule 1: Let $RP = (P, T; F, R_e, R_u, M_0)$ be a refined control flow, then a new refined control flow obtained by a sequential refinement of t in RP is $RP' = (P', T'; F', R'_e, R'_u, M_0)$, where

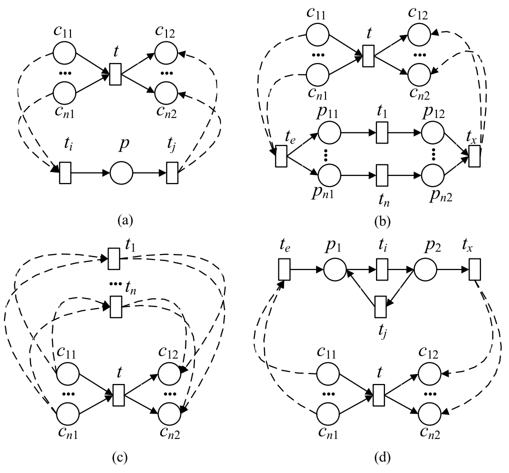

FIGURE 4. The refinement rules.

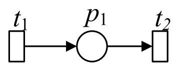

FIGURE 5. The sequence block for t in Fig. 3.

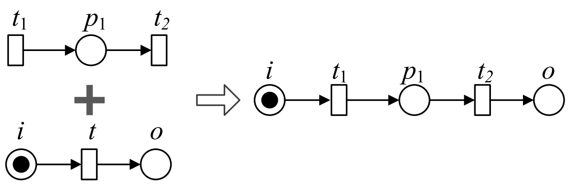

FIGURE 6. The refinement of the source control flow in Fig. 3.

(1) $P' = P \cup \{p\}$;
(2) $T' = T \cup \{t_i, t_j\} - \{t\}$;
(3) $F' = F \cup \{(c_{11}, t_i), \ldots, (c_{n1}, t_i), (t_j, c_{12}), \ldots, (t_j, c_{n2})\} - \{(c_{11}, t), \ldots, (c_{n1}, t), (t, c_{12}), \ldots, (t, c_{n2})\}$;
(4) $R'_e = R_e \cup \{t_i, t_j\} - \{t\}$;
(5) $R'_u = R_u$;
(6) M_0 is the initial marking, such that for any $p \in P, p \neq i$: $M_0(p) = 0$ and $M_0(i) = 1$.

Example 2: According to the scenario depicted in Figure 1, for *HarDesigner*, we can define the sequence block corresponding to the transition t in its source control flow, as shown in Figure 5.

Then, we refine the transition t in the source control flow in Figure 3 according to Rule 1, and the refined control flow obtained is shown in Figure 6.

The refinement rule based on concurrency blocks is depicted in Figure 4(b). Formally,

Rule 2: Let $RP = (P, T; F, R_e, R_u, M_0)$ be a refined control flow, then a new refined control flow obtained by a concurrent refinement of t in RP is $RP' = (P', T'; F', R'_e, R'_u, M_0)$, where

(1) $P' = P \cup \{p_{11}, p_{12}, \ldots, p_{n1}, p_{n2}\}$;
(2) $T' = T \cup \{t_1, \ldots, t_n, t_e, t_x\} - \{t\}$;

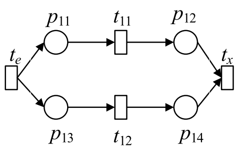

FIGURE 7. The concurrency block for t_1 in Fig. 6.

(3) $F' = F\cup\{(c_{11}, t_e), \ldots, (c_{n1}, t_e), (t_x,c_{12}), \ldots, (t_x,c_{n2})\} \cup\{(t_e, p_{11}), \ldots, (t_e, p_{n1}), (p_{11}, t_1), \ldots, (p_{n1}, t_n), (t_1, p_{12}), \ldots, (t_n, p_{n2}), (p_{12}, t_x), \ldots, (p_{n2}, t_x)\}-\{(c_{11}, t), \ldots, (c_{n1}, t), (t, c_{12}), \ldots, (t, c_{n2})\}$;
(4) $R'_e= R_e\cup\{t_i, t_j\}-\{t\}$;
(5) $R'_u= R_u\cup\{t_e, t_x\}$;
(6) M_0 is the initial marking, such that for any $p \in P, p \neq i$: $M_0(p) = 0$ and $M_0(i) = 1$.

Example 3: According to the scenario depicted in Figure 1, we can define a concurrency block for the transition t_1 in Figure 6, as shown in Figure 7.

Then, we refine the transition t_1 in Figure 6 according to Rule 2, and the refined control flow obtained is shown in Figure 8.

The refinement rule based on selection blocks is depicted in Figure 4(c). Formally,

Rule 3: Let $RP = (P, T; F, R_e, R_u, M_0)$ be a refined control flow, then a new refined control flow obtained by a selective refinement of t in RP is $RP' = (P', T'; F', R'_e, R'_u, M_0)$, where
(1) $P' = P$;
(2) $T' = T\cup\{t_1, \ldots, t_n\}-\{t\}$;
(3) $F' = F\cup\{(c_{11}, t_i), \ldots, (c_{n1}, t_i), (c_{11}, t_j), \ldots, (c_{n1}, t_j)\}\cup \{(t_i, c_{12}), \ldots, (t_i, p_{n2}), (t_j, c_{12}), \ldots, (t_j, p_{n2})\}-\{(c_{11}, t), \ldots, (c_{n1}, t), (t, c_{12}), \ldots, (t, c_{n2})\}$;
(4) $R'_e= R_e\cup\{t_1, \ldots, t_n\}-\{t\}$;
(5) $R'_u= R_u$.

The refinement rule based on iteration blocks is depicted in Figure 4(d). Formally,

Rule 4: Let $RP = (P, T; F, R_e, R_u, M_0)$ be a refined control flow, then a new refined control flow obtained by an iterative refinement of t in RP is $RP' = (P', T'; F', R'_e, R'_u, M_0)$, where
(1) $P' = P\cup\{p_1, p_2\}$;
(2) $T' = T\cup\{t_i, t_j, t_e, t_x\}-\{t\}$;
(3) $F' = F\cup\{(c_{11}, t_e), \ldots, (c_{n1}, t_e), (t_x,c_{12}), \ldots, (t_x,c_{n2})\}\cup \{(t_e, p_1), (p_1, t_i), (t_i, p_2), (p_2, t_j), (t_j, p_1), (p_2, t_x)\}- \{(c_{11}, t), \ldots, (c_{n1}, t), (t, c_{12}), \ldots, (t, c_{n2})\}$;
(4) $R'_e= R_e\cup\{t_i, t_j\}-\{t\}$;
(5) $R'_u= R_u\cup\{t_e, t_x\}$;
(6) M_0 is the initial marking, such that for any $p \in P, p \neq i$: $M_0(p) = 0$ and $M_0(i) = 1$.

Example 4: According to Figure 8, we finally obtain *HarDesigner*'s refined control flow as shown in Figure 9.

C. CONSTRUCTING PRIVATE PROCESSES

Based on the above refinement rules, we propose an approach for constructing private processes. In practice, our approach that serves as a modeling guideline can assists business

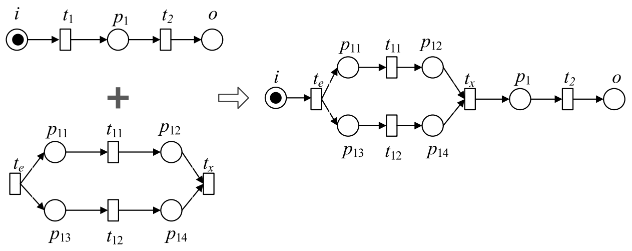

FIGURE 8. The refinement of t_1 via rule 2.

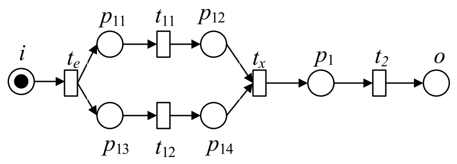

FIGURE 9. The refined control flow of *HarDesigner*.

designers reducing the modeling complexity of private processes. Concretely, our approach could be done by the following steps:
(1) Define a source control flow *SP* that savers as the start point for building the control flow of a particular private process.
(2) Analyze the refinable transition t in *SP*, and determine the basic block corresponding to t. Refine *SP* according to the corresponding refinement rule to generate a new refined control flow RP_1.
(3) If the granularity (the granularity is determined by business designers) of RP_1 is appropriate, then the refinement process ceases, otherwise jump step (4).
(4) Set RP=RP_1, and analyze the refinable transition $t \in RP.R_e$ in *SP*. Determine the basic block corresponding to t, and refine *SP* according to the corresponding refinement rule to produce a new refined control flow RP_2.
(5) Repeat step (4) until the appropriate granularity is reached, then a final refined control flow is obtained. Extract the places and transitions in the final refined control flow to construct a control flow named *Flow*.
(6) Associate each transition in *Flow* with a task to obtain a private process *BP*.

Example 5: According to the refined control flow shown in Figure 9, we only need to extract the places, transitions and flows in it, then we can construct the control flow of *HarDesigner*, as shown in Figure 10.

Then, according to the scenario depicted in Figure 1, we define the tasks in *HarDesigner*, as shown in Table 1. In Table 1, Name represents the name of the task; Org is the organization to which the task belongs; MsgRec denotes a set of messages which is received by the task before it is executed; MsgSend denotes a set of messages which is sent by the task after it is executed.

Finally, we define the mapping function $f = \{(t_{11}, task_{11}), (t_{12}, task_{12}), (t_2, task_2), (t_e, task), (t_x, task)\}$. The mapping function f associates each transition in the control flow shown in Figure 10 with a task, resulting in the private

TABLE 1. The tasks of *CPD*.

Task	Name	Org	MsgRec	MsgSend
*task*11	Design peripheral parts	HarDesigner	$\varnothing$	$\varnothing$
*task*12	Design motherboard	HarDesigner	$\varnothing$	{motPar}
*task*2	Integrate hardware and software	HarDesigner	{software}	$\varnothing$
task	Empty task	HarDesigner	$\varnothing$	$\varnothing$

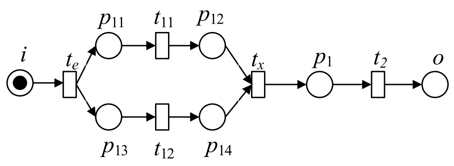

FIGURE 10. The control flow of *HarDesigner*.

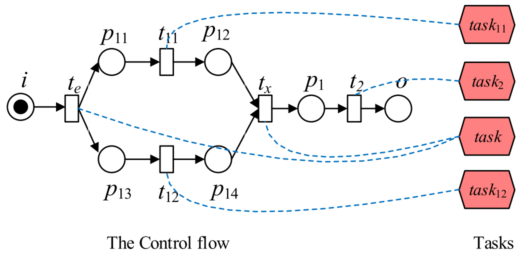

FIGURE 11. The private process of *HarDesigner*.

process of *HarDesigner*, as shown in Figure 11. In particular, transitions t_e and t_x are only associated with empty tasks for routing purposes.

Following the above procedure, we can analyze the correctness of private processes. Generally, the correctness of private processes can be analyzed at both the syntax and semantic levels [7]. Therefore, we distinguish between two kinds of correctness in our context: *syntax correctness* and *semantic correctness*.

Compared with traditional modeling languages of business processes such as EPC, YAWL and BPMN, WF-nets have a more mathematical representation and are more easily converted into other specific modeling languages such as BPEL in practice. Therefore, in this paper we define syntax correctness as the fact that the control flow between tasks in private processes is a WF-net. That is, let $BP=$ (*Tasks*, *Flow*, f) be a private process, BP is syntactically correct, if and only if *Flow* is a WF-net.

By analyzing the refinement procedure of private processes, we can conclude that the final refined control flow is a WF-net.

Theorem 1: Let $RP = (P, T; F, R_e, R_u, M_0)$ be a final refined control flow, then RP is a WF-net.

Proof: We prove this theorem by induction on n, where n refers to the number of refinements. Let SP be the source control flow of RP.

(1) For $n = 1$, we need to prove that SP is a WF-net. By Definition 9, we know that the conclusion holds.

(2) For $n = k$, the refined control flow RP_k is a WF-net, where RP_k is generated by refining SP for k times.

(3) For $n = k+1$, we need to prove that the refined control flow RP_{k+1} is a WF-net, where RP_{k+1} is generated by refining SP for $(k+1)$ times.

   (1) If RP_{k+1} is obtained by a sequential refinement of t in RP_k, then we know that ${}^\bullet i = \emptyset$ and $o^\bullet = \emptyset$ in RP_{k+1} according to Rule 1. According to Definition 5, we know that $A_{e1} = \{t_i\}$, $A_{x1} = \{t_j\}$, $F = \{(t_i, p), (p, t_j)\}$, and then derive t_i and t_j are connected. Since RP_k is a WF-net, we know that i and t are connected and t and o are connected, and then derive i and t_i are connected, t_i and t_j are connected, and t_j and o are connected. If an extra transition e is added into RP_{k+1}, such that $e^\bullet = o$ and ${}^\bullet e = i$, then we can derive that RP_{k+1} is strongly connected.

   (2) If RP_{k+1} is obtained by a concurrent refinement, or a selective refinement, or an iterative refinement of t in RP_k, the proof is similar to (a). To save space, the rest of the proof is omitted.

Combining the analysis of (1), (2) and (3), we can derive that RP is a WF-net. □

Specially, for a refined control flow $RP= (P, T; F, R_e, R_u, M_0)$, and two elements e_1 and $e_2 \subseteq P \cup T$, if e_1 and e_2 are connected, if and only if the condition $\{(e_1, p_1), \ldots, (p_n, e_2)\} \subseteq F$ holds, where for any $i \in [1..n]$, $p_i \subseteq P \cup T$. The term strongly connected means that for any two elements e_1 and $e_2 \subseteq P \cup T$, e_1, e_2 are connected.

Theorem 1 states that the final refined control flow is a WF-net. Therefore, we can conclude that the built private processes are syntactically correct.

Corollary 1: Let BP be a built private process, then BP is syntactically correct.

Proof: Let $RP= (P, T; F, R_e, R_u, M_0)$ be a final refined control flow. Thus, we can construct the control flow of BP, i.e., *Flow*$= (P, T; F, M_0, i, o)$. According to the modeling approach described in section 4.2, we know that each transition in *Flow* is associated with a task, i.e., there exists a mapping function f, such that for any $t \in T$, there is a task *task*, such that $f(t) =$ *task*.

By Theorem 1, we know that RP is a WF-net, and then derive that *Flow* is a WF-net as well. Hence, we know that the structure between the tasks in BP (determined by *Flow*) meets the properties of WF-nets. □

Example 6: According to Corollary 1, we know that *HarDesigner*'s private process is syntactically correct, because the control flow shown in Figure 11 has 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 control flow is strongly connected.

As the motivation example illustrates, the correctness of private processes (such as *HarDesigner*) has a direct impact on the correctness of collaborative business processes (such as *CPD*). To improve the efficiency of formal verification of collaborative business processes, it is usually required that the private process in the collaboration is correct first.

Informally, semantic correctness means that private processes are able to successfully terminate through the execution of sequences of tasks. Since the private process is a single process, its semantic correctness can be defined with the sound property.

Definition 12 (Semantic Correctness): A business process *BP* is semantically correct, if and only if the following conditions hold:

(1) $\forall M$: $M_0[\sigma_1 > M \rightarrow M[\sigma_2 > M_f$, where M_f is the final marking of *BP*, such that for any $p \in BP.Flow.P, p \neq o$: $M_f(p) = 0$ and $M_f(o) = 1$;

(2) $\forall M'$: $M_0[\sigma > M' \wedge M' \geq o \rightarrow M' = o$;

(3) $\forall task \in BP.Task*$, there exist two markings M and M', such that $M_0[\sigma > M[task > M'$.

where $\sigma, \sigma_1, \sigma_2 \in BP.Task*$; $M[\sigma > M'$ indicates that *BP* reaches a new marking M' from the marking M by executing a sequence of tasks σ.

In Definition 12, (1) refers to any marking M that is reachable from the initial marking, the final marking can be reachable from M by the execution of a sequence of tasks; (2) states that if the final marking is reached, then there is exactly one token in the place o, and no tokens in the other places. (1) and (2) limit the situation where deadlocks and livelocks do not occur in *BP*. (3) states that there are no dead tasks in *BP*.

In particular, Definition 12 and Definition 4 are similar as they are both defined with soundness. Yet, Definition 4 only defines the correctness of the control flow of private processes from the viewpoint of transitions, Definition 12 defines the semantic correctness of private processes from the viewpoint of tasks, and involves both the control flow and the message flow (i.e., the interaction between private processes via message exchange).

To prove the private process built through our approach is semantically correct, we first present several theorems related to the basic blocks in the following.

Theorem 2: Let t be a refinable transition, and t is refined by a sequence block $B = (P, T; F, A_e, A_x, R_e, R_u)$. If $M[t > M'$, then $M[B > M'$.

Proof: Since $M[t > M'$, we know that t_i is enable under M according to Definition 6 and Rule 1, and then derive that $M[t_i > \{p\}, \{p\}[t_j > M'$. □

Theorem 3: Let t be a refinable transition, and t is refined by a concurrency block $B = (P, T; F, A_e, A_x, R_e, R_u)$. If $M[t > M'$, then $M[B > M'$.

Proof: Since $M[t > M'$, we know that t_e is enable under M according to Definition 7 and Rule 2, and then derive that $M[t_e > \{p_{11}, \ldots, p_{n1}\}$, $\{p_{11}, \ldots, p_{n1}\}[\{t_1, \ldots, t_n\} > \{p_{12}, \ldots, p_{n2}\}$, $\{p_{12}, \ldots, p_{n2}\}[t_j > M'$. □

Theorem 4: Let t be a refinable transition, and t is refined by a selection block $B = (P, T; F, A_e, A_x, R_e, R_u)$. If $M[t > M'$, then $M[B > M'$.

Proof: Since $M[t > M'$, we know that for any transition $t \in \{t_1, \ldots, t_n\}$ is enable under M according to Definition 8 and Rule 3, and then derive that $M[\{t_1, \ldots, t_n\} > M'$. □

Theorem 5: Let t be a refinable transition, and t is refined by an iteration block $B = (P, T; F, A_e, A_x, R_e, R_u)$. If $M[t > M'$, then $M[B > M'$.

Proof: Since $M[t > M'$, we know that t_e is enable under M according to Definition 9 and Rule 4, and then derive that $M[t_e > \{p_1\}$, $\{p_1\}[t_i > \{p_2\}$, $\{p_2\}[t_j > \{p_1\}, \ldots,$ $\{p_1\}[t_i > \{p_2\}$, $\{p_2\}[t_x > M'$. □

By the use of Theorem 2∼ Theorem 5, we can derive that a final refined control flow is sound [7].

Theorem 6: Let $RP = (P, T; F, R_e, R_u, M_0)$ be a final refined control flow, then RP is sound.

Proof: We prove this theorem by induction on n, where n refers to the number of refinements. Let *SP* be the source control flow of *RP*.

(1) For $n = 1$, we need to prove that *SP* is sound. According to Definition 4 and Definition 10, we know that $M_0[t > M_f$, and then derive that *RP* is sound.

(2) For $n = k$, the refined control flow RP_k is sound, where RP_k is generated by refining *SP* for k times.

(3) For $n = k+1$, we need to prove that the refined control flowRP_{k+1} is sound, where RP_{k+1} is generated by refining *SP* for $(k+1)$ times.

  (1) Let t be a refinable transition in *RP*, and t is refined by a sequence block B. Since RP_k is sound, we know that $M[t > M'$. According to Theorem 2, we know that $M[B > M'$, and then derive that RP_{k+1} is sound.

  (2) Let t be a refinable transition in *RP*, and t is refined by a concurrency block B. Since RP_k is sound, we know that $M[t > M'$. According to Theorem 3, we know that $M[B > M'$, and then derive that RP_{k+1} is sound.

  (3) Let t be a refinable transition in *RP*, and t is refined by a selection block B. Since RP_k is sound, we know that $M[t > M'$. According to Theorem 4, we know that $M[B > M'$, and then derive that RP_{k+1} is sound.

  (4) Let t be a refinable transition in *RP*, and t is refined by an iteration block B. Since RP_k is sound, we know that $M[t > M'$. According to Theorem 5, we know that $M[B > M'$, and then derive that RP_{k+1} is sound.

Combining the analysis of (1), (2) and (3), we can derive that *RP* is sound. □

With Definition 12 and Theorem 6, we can prove the private process built by our approach is semantically correct.

Corollary 2: Let *BP* be a private process built by our approach, then *BP* is semantically correct.

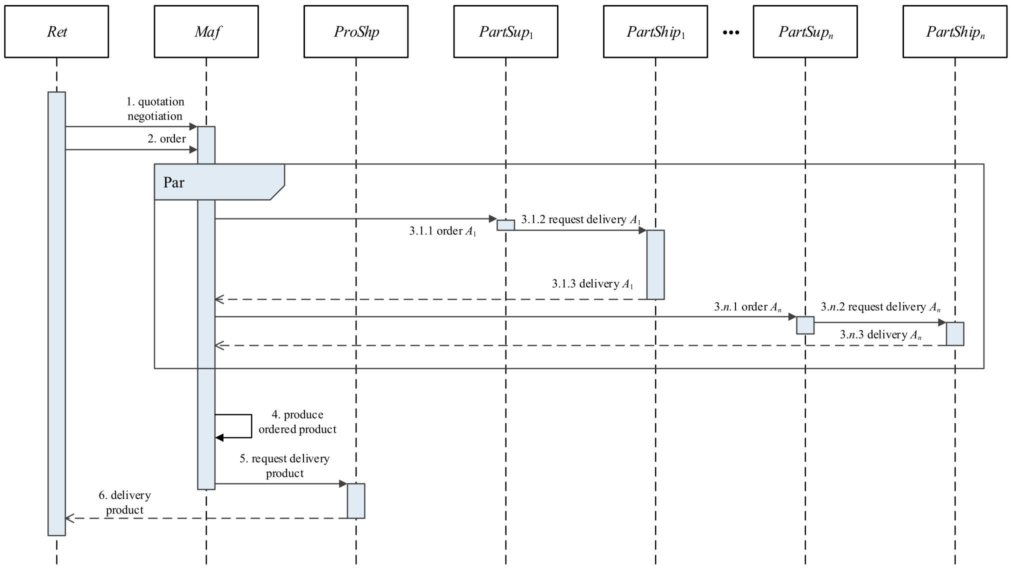

FIGURE 12. **The interaction of *OP* depicted by UML sequence diagram.**

Proof: Let$RP = (P, T; F, R_e, R_u, M_0)$ be the final refined control flow of *BP*. We can construct the control flow of *BP*, i.e.,*Flow*$= (P, T; F, M_0, i, o)$. According to the modeling approach presented in section 4.2, we know that each transition in *Flow* is associated with a task, i.e., there exists a mapping function f, such that for any $t \in T$, there is a task *task*, such that $f(t)$ =*task*. Using *Flow* and f, we construct the private process *BP*=(*Tasks*, *Flow*, f), where *Tasks* =$\{task|\ task= f(t)\}$.

According to Theorem 6, we know that *RP* is sound, and then derive that *Flow* is sound as well. That is,

(1) $\forall M: M_0[\sigma_1 > M \rightarrow M[\sigma_2 > M_f$;
(2) $\forall M': M_0[\sigma > M' \wedge M' \geq o \rightarrow M' = o$;
(3) $\forall t \in Flow.T$, there exist two markings $nnnM, M'$ such that $M_0[\sigma > M[t > M'$.

where $\sigma, \sigma_1, \sigma_2 \in Flow.T*$; $M[\sigma > M'$ indicates that *Flow* reaches a new marking M' from the marking M by executing a sequence of transitions σ.

We recall that the execution of private processes under a perfect environment E. This means that E is always able to send or receive a message whenever a private process is expected to receive or send such a message. Therefore, we can derive the following conditions hold:

(1) $\forall M: M_0[f(\sigma_1) > M \rightarrow M[f(\sigma_2) > M_f$;
(2) $\forall M': M_0[f(\sigma) > M' \wedge M' \geq o \rightarrow M' = o$;
(3) $\forall task \in BP.Task$, there exist two markings nM and M' such that $M_0[f(\sigma) > M[task > M'$.

where σ denotes a sequence of transitions, while $f(\sigma)$ denotes the sequence of tasks corresponding to σ.

According to Definition 12, we derive that *BP* is semantically correct. □

Corollary 2 states that the private process built via our approach is semantically correct. Thus, a subsequent correctness analysis for private processes is avoided, and the efficiency of the correctness analysis of collaborative business processes is improved eventually.

Example 7: According to Corollary 2, we know that *HarDesigner*'s private process is semantically correct. That is, since the control flow shown in Figure 11 is sound and the message *software* received by the task $task_2$ is available, we can derive that the private process must be executed correctly (according to definition 3) without any undesired outcome, such as deadlocks and livelocks.

V. CASE STUDY

In this section, we first choose a simplified order process (*OP*) from the supply chain collaboration domain as our case study. Then, we illustrate the effectiveness of our approach via modeling the private processes in *OP*. Finally, we quantitatively evaluate the analytical efficiency of our approach and the related work. All experiments were carried out on a PC with 1.60GHz Processor and 8GB of RAM, running Windows 10.

A. ORDER PROCESS

OP involves $2n + 3$ participants, i.e., 1 retailer (denoted as *Ret*), 1 manufacturer (denoted as*Maf*), n parts suppliers (denoted as$PartSup_1, \ldots, PartSup_n$), n parts shippers (denoted as$PartShp_1, \ldots, PartShp_n$) and 1 product shipper (denoted as*ProShp*). The interaction between these participants is depicted in Figure 12, which includes the following steps:

(1) *Ret* negotiates with *Maf* on the price of the ordered product;
(2) After the price is agreed, *Ret* submits an order O to *Maf*;
(3) After receiving O, *Maf* analyzes O, and determines that O consists of parts $A_1 \sim A_n$, and then sends requests for purchasing these parts to $PartSup_1, \ldots, PartSup_n$;

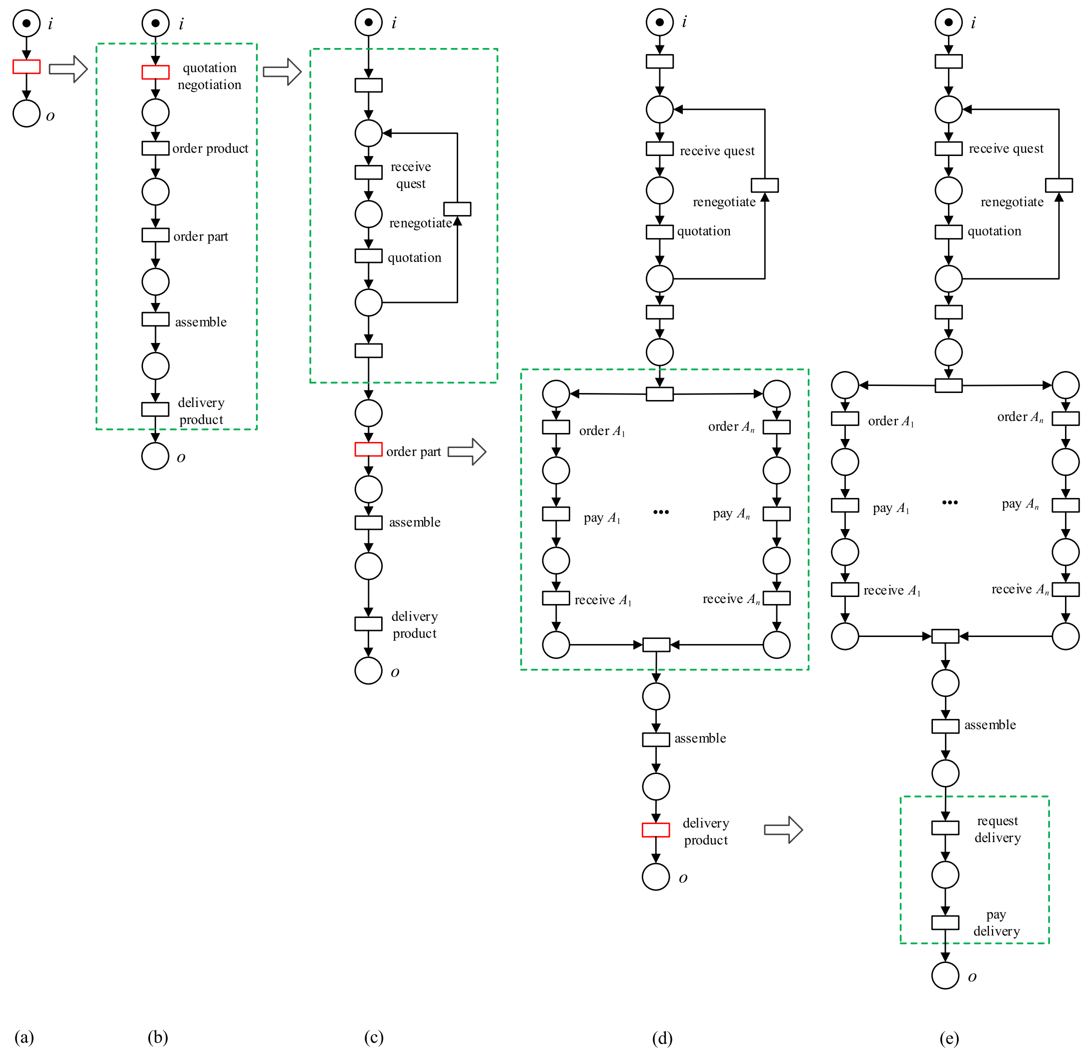

FIGURE 13. **The construction of *Maf*'s private process.**

(4) After receiving the purchasing requests, $PartSup_1$, …, $PartSup_n$ will notify $PartShp_1$, …, $PartShp_n$ to deliver the purchasing parts to *Maf*;

(5) After receiving these parts, *Maf* produces an ordered product *P* by assembling these parts;

(6) *Maf* notifies *ProShp* to deliver *P* to *Ret*.

B. MODELING THE PRIVATE PROCESSES IN OP

In practice, the private processes involved in *OP* can be constructed by our approach. To save space, we only present the procedure for constructing*Maf*'s private process. The private processes of other participants in *OP* can be built in a manner similar to *Maf*.

Concretely, we first define a source control flow shown in Figure 13(a) that savers as the start point for constructing the control flow of *Maf*'s private process. Afterwards, we repeatedly employ Rule 1∼ Rule 4 to refine the refinable transitions in the refined control flows shown in Figure 13 (a)∼ Figure 13(d) to obtain the control flow (e) of *Maf*, as shown in Figure 13 (e). For example, the refinement of the transition *quotation negotiation* in Figure 13(b) is achieved by using Rule 4 and Rule 1. At last, we associate each transition in the control flow of *Maf* with a specific task to construct the private process of *Maf*.

From the procedure shown in Figure 13, we can see that our approach conforms to business designers' thinking habits during the construction of private processes. Therefore, in practice our approach that serves as a modeling guideline can assist business designers reducing the modeling complexity of private processes.

C. EFFICIENCY EVALUATION

In order to evaluate the analysis efficiency of our approach, we choose representative research [6], [7] for experiments. The basic idea of the approaches in [6] and [7] remains the same. Thus, we regard them as an approach in our context. Additionally, in this paper we focus on structured private processes (the reasons are illustrated in section 2).

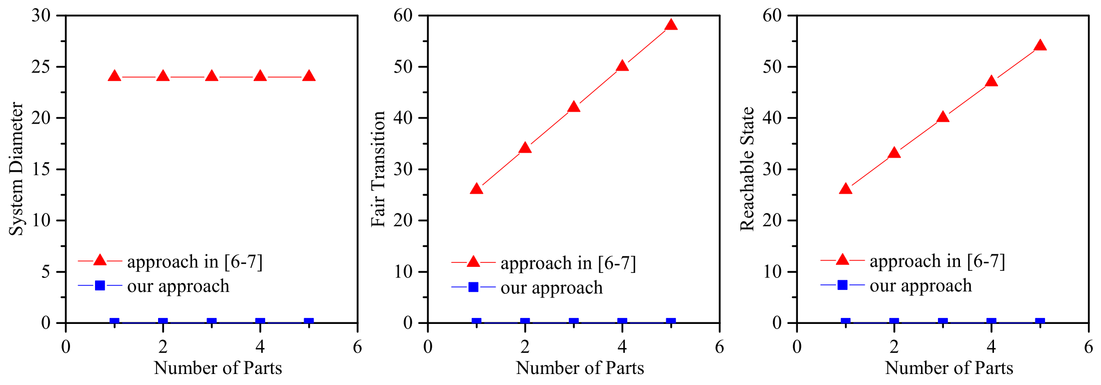

FIGURE 14. Analysis efficiency of syntax correctness per approach.

Thus, these structured private processes can be abstracted into four basic blocks in an easy way. Based on these four basic blocks, our approach is able to construct private processes with correctness using the presented refinement rules. Since the constructed private processes are correct at both the syntax and semantic levels, an extra effort of correctness verification for private processes is avoided. In contrast, for modeling flexibility, the approach in [6] and [7] does not consider correctness when constructing private processes. Thus, the constructed models are usually unstructured and may be incorrect. In order to ensure the correctness of these unstructured processes, the work in [6] and [7] proposed methods for verifying correctness. Meanwhile, both our work and the work in [6] and [7] use *soundness* as the correctness criterion of private processes. Hence, based on the above arguments, we claim that it is reasonable to compare the analysis efficiency of our approach and the approach [6], [7].

Concretely, we take *Maf*'s private process shown in Figure 13(e) as an experimental object, and then conduct a series of experiments on the object to quantitatively evaluate our approach and the approach in [6] and [7] in terms of analysis efficiency. In our work, we employ model checking to as a formal method for syntactic and semantic correctness analysis. The basic idea of model checking is to represent the system and the properties to be verified as a state transition model M and a temporal logic formula ϕ, respectively. Then, it converts "whether the system satisfies the desired properties" into "whether M satisfies the formula ϕ, i.e., $M| = \phi''$. Since the efficiency of model checking is determined by the state space of M, we can use the metrics in [19], i.e., reachable state, fair transition and system diameter, to evaluate the analytical efficiency of our approach and the approach in [6] and [7]. System diameter is the search depth, indicating the distance from the initial state to the farthest reachable state, while reachable state and fair transition reflect the size of the state space of the system and directly affect the efficiency of model checking [19]. Particularly, reachable state, fair transition and system diameter are automatically calculated in the experiment. Specifically, given a state transition model M, the number of the states and transitions in M can be used to represent reachable state and fair transition, respectively. Additionally, system diameter can be calculated using the standard DFS algorithm [26], which expresses the length of the longest path that can be reached from the initial state of M.

Note that in this paper, we only consider key factors (i.e., reachable state, fair transition, and system diameter) that affect the efficiency of formal verification, and use these factors as metrics to evaluate the analysis efficiency of the correctness of private processes. How to use model checking to verify the correctness of private processes is out of scope of this paper, and interesting readers are referred to [20].

In our context, we distinguish between two kinds of analysis efficiency: the analysis efficiency of syntax correctness and the analysis efficiency of semantic correctness.

The analysis efficiency of syntax correctness refers to the cost of building a private process with syntax correctness. Concretely, given a private process P generated by refinement, we abstract the places and transitions in P into nodes, and then abstract the flows in P into edges to generate a directed graph G. Assuming that the number of nodes in G and the number of edges in G are V and E, respectively. The length of the longest path in G is L. Then, the cost of building private process P with syntax correctness is represented by the reachable state $S = V$, the fair transition $T = E$, and the system diameter $D = L$. Among them, the longest path in G can be obtained by the DFS algorithm [26].

For our approach and the approach in [6] and [7], the comparison of their analysis efficiency in terms of syntax correctness is depicted in Figure 14. In Figure 14, we can see that as the number of purchasing parts increases (n increases from 1 to 5), the reachable state and fair transition of the state model M corresponding to [6] and [7] increases linearly, and the system diameter of M remains unchanged. In contrast, the reachable state, fair transition and system diameter corresponding to our approach are all zero. Hence, we can conclude that since our approach can ensure syntax correctness in advance and does not need to traverse the control flow of *Maf*'s private process, the analysis efficiency

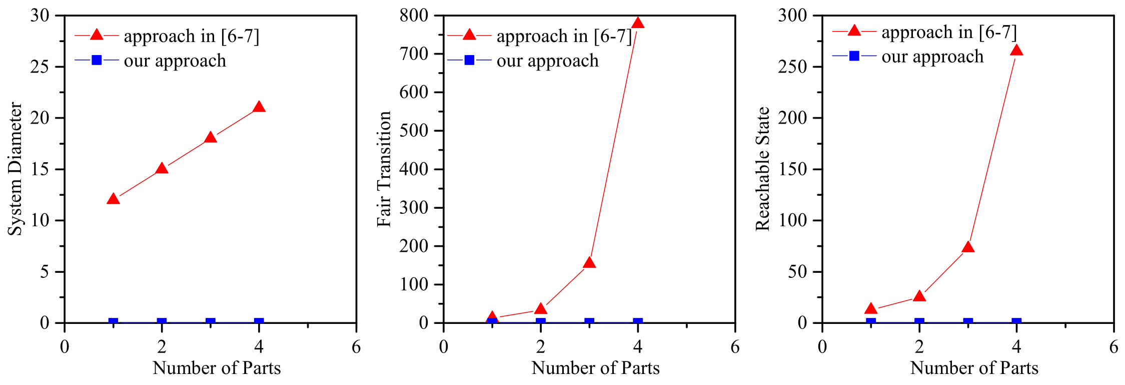

FIGURE 15. Analysis efficiency of semantic correctness per approach.

TABLE 2. Experimental results in terms of syntax correctness.

Case	Size	The approach in [6-7]	Our approach
	|*P*|/|*T*|	*RS* | *FT* | *SD*	*RS* | *FT* | *SD*
Ca-09	27/30	57 | 66 | 22	- | - | -
Ca-13	39/32	71 | 84 | 29	- | - | -
Ca-24	14/13	27 | 28 | 20	- | - | -
Ca-30	42/31	73 | 88 | 22	- | - | -
Ca-36	33/25	58 | 62 | 38	- | - | -
Ca-45	24/22	46 | 52 | 26	- | - | -
Ca-56	14/11	25 | 26 | 18	- | - | -

of syntax correctness of our method obviously outperforms the approach in [6] and [7].

The analysis efficiency of semantic correctness refers to the cost of building a private process with semantic correctness. Concretely, given a private process P built by refinement, we generate the reachable graph G of P. Assuming that the number of nodes in G and the number of edges in G are V and E, respectively. The length of the longest path in G is L. Then, the cost of building private process P with semantic correctness is represented by the reachable state $S = V$, the fair transition $T = E$, and the system diameter $D = L$. Among them, the longest path in G can be obtained by the DFS algorithm [26], and the reachable graph can be generated using the algorithm described in [25].

For our approach and the approach in [6] and [7], the comparison of their analysis efficiency in terms of semantic correctness is depicted in Figure 15. In Figure 15, we can see that as the number of purchasing parts increases (n increases from 1 to 5), the system diameter of the state model M corresponding to [6] and [7] increases linearly, while the reachable state and fair transition of M increases exponentially. In particular, when $n = 5$, the state model corresponding to [6] and [7] cannot be generated. This makes *Maf*'s semantic correctness undecidable, and this in turn results in the correctness of *OP* undecidable as well. In contrast, the reachable state, fair transition and system diameter corresponding to our approach are all zero. Hence, we can conclude that since our method can ensure semantic correctness in advance and does not need to exploit and analyze the reachable graph of *Maf*'s private process, the analysis efficiency of semantic correctness of our approach significantly outperforms the approach in [6] and [7]. Note that the state model M here refers to the reachable graph generated by *Maf*'s private process. Several algorithms have been presented to generate the reachable graph of Petri nets, and interesting readers are referred to [21].

Currently, there are no public collaborative business processes available for experiments [27]. Yet, in order to evaluate the analytical effectiveness of our approach more generally, we choose 60 real-world cases (such as a booking system, a risk management process, and an emergency management system) from existing research papers and the BPMN case base (http://www.bpmn.org/) for experiments. These cases specify actual scenarios in different areas, which represent diverse and practical private processes.

For these sixty cases, we first analyze syntax correctness, and Table 2 presents the experimental results of some cases. In Table 2, $|P|$ and $|T|$ represent the number of places and transitions in the case, respectively. *RS*, *FT*, and *SD* represent the reachable state, fair transition, and system diameter used to identify the analytical efficiency of syntax correctness, respectively.

In Table 2, we can see that as the structural complexity of the private process increases, the reachable state, fair transition and system diameter of the state model M corresponding to [6] and [7] increases as well. For example, the structure of *Ca*-13 is more complex than *Ca*-9, i.e., the number of nodes (places and transitions) and flows in *Ca*-13 is more than *Ca*-9, and hence the analytical efficiency of the syntax correctness of *Ca*-13 is lower than *Ca*-9. In contrast, since our

TABLE 3. Experimental results in terms of semantic correctness.

Case	Size	The approach in [6-7]	Our approach
	$\|P\|/\|T\|$	*RS* | *FT* | *SD*	*RS* | *FT* | *SD*
Ca-09	27/30	96 | 238 | 17	- | - | -
Ca-13	39/32	318 | 883 | 18	- | - | -
Ca-24	14/13	144 | 321 | 24	- | - | -
Ca-30	42/31	88 | 173 | 21	- | - | -
Ca-36	33/25	150 | 326 | 23	- | - | -
Ca-45	24/22	26 | 41 | 16	- | - | -
Ca-56	14/11	16 | 20 | 11	- | - | -

approach can ensure syntax correctness in advance and does not need to traverse the control flow of the private process (i.e., *RS*, *FT* and *SD* are '−'), the analytical efficiency of the syntax correctness of our approach obviously outperforms the approach in [6] and [7].

Afterwards, we analyze semantic correctness, and Table 3 presents the experimental results of some cases. Note that $|P|$, $|T|$, *RS*, *FT*, and *SD* are specified in Table 2.

In Table 3, we can see that as the behavioral complexity of the private process increases, the reachable state, fair transition and system diameter of the state model M corresponding to [6] and [7] increases rapidly. For example, the behavioral structure of *Ca*-13 is more complex than *Ca*-24, i.e., the number of nodes and transitions in the reachable graph of *Ca*-13 is significantly more than *Ca*-24, and hence the analytical efficiency of the semantic correctness of *Ca*-13 is much lower than *Ca*-9. In contrast, since our approach can ensure semantic correctness in advance and does not need to exploit and analyze the reachable graph of the private process (i.e., *RS*, *FT* and *SD* are 'X'), the analysis efficiency of semantic correctness of our method significantly outperforms the approach in [6] and [7].

VI. RELATED WORK

Our work is related to two research axes: (1) modeling collaborative business processes; and (2) the correctness analysis of collaborative business processes.

A. MODELING COLLABORATIVE BUSINESS PROCESSES

The existing work in this area can be divided into two categories: the top-down modeling approach and the bottom-up modeling approach.

1) THE TOP-DOWN MODELING APPROACH

Van der Aalst *et al.* [9] presented a Public-To-Private (P2P) approach that relies on rules based on behavioral inheritance to model inter-organizational workflows. Their approach creates a shared public workflow (i.e., contract), and then partition the public workflow over organizations. At the end, a private workflow is created for each party that corresponds to a subclass of the respective part of the public workflow in terms of mapping rules. After that, Aalst *et al.* [10] used a contract to describe the composition of public views in the collaboration, and then each organization incrementally implements its own part involved in the contract using transformation rules. Matthias *et al.* [28] presented a framework called VerChor to construct the collaborative business process in a top-down manner. Their approach first defines a choreography that specifies the interaction between peers from a global point of view. Afterwards, the choreography is used to obtain peers via projection while preserving realizability. That is, the composition of these peers conforms to the choreography. At last, the developer may implement the peers by transformation rules, or adding business code. Marco *et al.* [29] presented an approach that constructs the collaborative business process by reusing existing third-party services. Their approach first specifies a choreography that model the external interaction between participating services by specifying peer-to-peer message exchanges from a global point of view. Then, a set of services is selected from the service inventory according to the peers generated from the choreography. At Last, the developer composes these selected services to achieve the choreography via coordination delegates. Coordination delegates are additional software entities that are used to control the interaction between services to achieve choreography realizability enforcement.

The above approaches either focus on the design of transformation rules, or deal with the realizability problem with respect to choreographies. However, the problem of the correctness of collaborative business processes is not considered.

2) THE BOTTOM-UP MODELING APPROACH

Wang *et al.* [8] presented an approach for the construction of collaborative business processes considering privacy. Their approach first assumes that the internal view (i.e., private process) of participating organizations in the collaboration are extend free choice nets and presents a set of meta constructs including sequence, choice, concurrency and loop meta constructs. Afterwards, the approach proposes reductive rules for these meta constructs to obtain public views corresponding to internal views. At last, by composing these public views, a collaborative business process is built. Eshuis *et al.* [14] introduced the notion of process view. A process view shields secret or irrelevant details from a private business process by projection rules (such as Black box projection, Glass box projection, Gray box projection), thus allowing an organization to reveal only public and relevant parts of its private business process to partner organizations. By composing the process view of each party in the collaboration, a collaborative business process can be obtained. Fei *et al.* [15] presented an approach that relies on Petri nets and Pi calculus for modeling collaborative business processes. Their approach employs Petri nets to model the control flow of private processes in the collaboration, and uses Pi calculus to specify the interaction between private processes. Then, the approach generates the communication behavior of each participating organization using the reachable graph of Petri nets. Lastly, by composing the communication behavior of these participating organizations using parallel composition, a collaborative business process can be obtained. Qi *et al.* [11] proposed an approach for constructing collaborative business processes by extracting public processes from private processes. Their approach defines the business process model to represent private processes, and abstracts it into four basic blocks. Based on these four basic blocks, this approach proposes extraction rules for these basic blocks to obtain the public process of each party and composes these public processes to establish a collaborative business process considering privacy. However, the above approaches mainly focus on the construction of collaborative business processes with individual features (e.g., autonomy and privacy), but fail to consider the correctness of collaborative business processes.

B. CORRECTNESS ANALYSIS OF COLLABORATIVE BUSINESS PROCESSES

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

1) AUTOMATA-BASED APPROACHES

Zhou *et al.* [30] 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.* [31] 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 [7] 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) [13] 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.* [16] 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 [7]. 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.* [12] 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. Zeng *et al.* [6] 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 reachable 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.* [32] 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.* [33] 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, Bi *et al.* [34] 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 abnormity. Lastly, an optimal controller is constructed for the administrator of service composition to avoid deadlocks in service choreography.

3) PROCESS ALGEBRA-BASED APPROACHES

Wong *et al.* [35] 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.* [36] 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.* [37] 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 composite correctness, assuming that self-correctness is satisfied in advance. In fact, if the structure of private processes is complex, then the state-space explosion problem may occur. This problem will directly affect the correctness analysis of private processes, and eventually affect the correctness analysis of collaborative business processes. How to avoid state-space explosion in the correctness analysis of private processes to improve the correctness analysis of collaborative business process is a key problem yet the above approaches have not discussed this problem in detail.

4) OTHER APPROACHES

Sheng *et al.* [38] proposed an automata-based approach for behavioral modeling and automated verification of web service-based processes. Their approach models operational and control behaviors of web service-based processes in terms of automata, and then proposes an automated verification approach based on symbolic model checking. In particular, the approach extracts the checking properties, in the form of temporal logic formulas, from control behaviors, and automatically verifies the properties in operational behaviors using the NuSMV model checker. To verify BPMN processes with a large model size, Dechsupa *et al.* [39] presented an approach for transforming the BPMN process into a colored petri net using Partitioning. Their approach proposes a framework for transforming BPMN processes into colored Petri nets from the control flow and data flow perspectives. Particularly, the partitioning technique for BPMN processes is applied to reduce the complexity and leads to a CPN model that can support the hierarchical and compositional verification techniques that are suitable for the large-scale BPMN design models. Afterwards, correctness properties can be defined and automatically verified using CPN tools. Liu *et al.* [40] proposed an extended logical petri nets (ELPNs) to model and analyze business processes. Their approach mainly illustrates the concept, firing rules, and reachable graph construction algorithm of ELPNs. Taking advantage of the reachable graph, the business processes built by ELPNs can be verified. Typically, private processes under a cross-organizational setting involve both the control flow and the message flow (i.e., the interaction between private processes via message exchange). However, the above approaches only focus on the business process within a single organization, and deal with the correctness of the control flow of business processes. Additionally, these approaches ensure correctness by verification, yet our approach can guarantee the fact that the built private processes itself is correct, and thus a subsequent correctness verification is avoided.

A comparative summary of previous efforts in this area is given in Table 4. The columns of the table correspond to the following criteria.

1) **CC** indicates whether the approach supports composite correctness fully (+), partially (+/−) or not at all (−). It can be seen that the existing approaches mainly focus on composite correctness, but fail to consider self-correctness.
2) **SC** indicates whether the approach supports self-correctness fully (+), partially (+/−) or not at all (−). It can be seen that only the approaches in [38]–[40] focus on self-correctness, yet the message flow is ignored.

TABLE 4. A comparison of related work on correctness analysis.

	CC	SC	MDC
Wang et al. [8]	-	-	N
Aalst et al. [9-10]	-	-	N
Mo et al. [11]	-	-	N
Eshuis et al. [14]	-	-	N
Dai et al. [15]	-	-	N
Matthias et at. [28]	-	-	N
Marco et al. [29]	-	-	N
Zeng et al. [6]	+	-	N
Aalst et al. [7]	+	-	V
Ge et al. [12]	+	-	V
Zhang et al. [16]	+	-	V
Zhou et al. [30]	+	-	V
Flavio et al. [31]	+	-	V
Kheldoun et al. [32]	+	-	V
Du et al. [33]	+	-	V
Bi et al. [34]	+	-	V
Wong et al. [35]	+	-	V
Mendoza et al. [36]	+	-	V
Zhu et al. [37]	+	-	V
Sheng et al. [38]	-	+/-	V
Dechsupa et al. [39]	-	+/-	V
Liu et al. [40]	-	+/-	V
Our approach	-	+	C

3) **MDC** indicates the means for determining correctness, where N for no means provided to determine correctness, V for determining correctness by verification, and C for determining correctness by construction, i.e., the constructed private process itself is correct. It can be seen that only our approach can product private processes with correctness, thus avoiding a subsequent correctness verification.

VII. CONCLUSION

In the bottom-up approach, private processes are the basis for constructing collaborative business processes. Their correctness is considered to be an important issue in the construction of private processes, and affects the correctness analysis of collaborative business processes. In order to effectively construct private processes, we propose an approach based on four basic blocks to stepwise construct private processes. The approach that serves as a modeling guideline can assist business designers reducing the modeling complexity of private processes. Meanwhile, it can ensure that the constructed private processes are correct at both the syntax and semantic levels, and thus a subsequent correctness analysis for private processes is avoided.

The future work will be mainly carried out in the following three aspects: 1) since our approach focuses on self-correctness, i.e., assuming a perfect environment exists. In fact, if we relax this condition and consider the execution of tasks under an inter-organizational environment, then composite correctness is involved. The existing work on the verification of composite correctness usually suffers from the state-space explosion problem, how to effectively verify composite correctness will be discussed in our future work; and 2) In this paper, we only focus on message-based asynchronous communication between private processes. The future work will consider self-correctness in the case involving both asynchronous and synchronous communication.

REFERENCES

[1] L. Wen, J. Wang, W. M. P. van der Aalst, B. Huang, and J. Sun, "Mining process models with prime invisible tasks," *Data Knowl. Eng.*, vol. 69, no. 10, pp. 999–1021, 2010.

[2] A. Sill, "Cloud, data, and business process standards for manufacturing," *IEEE Cloud Comput.*, vol. 3, no. 4, pp. 74–80, Jul./Aug. 2016.

[3] 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.

[4] L. Ying, 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, 2017.

[5] 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, Apr. 2016.

[6] 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.

[7] W. Aalst, "Modeling and analyzing interorganizational work-flows," in *Proc. 1st Int. Conf. Appl. Concurrency Syst. Design*, Los Alamitos, CA, USA, 1998, pp. 262–272.

[8] J. Wang, H. Hu, P. Yu, J. Lv, and J. Ge, "Public view and object Petri net based modeling of cross-organizational process," *J. Frontiers Comput. Sci. Technol.*, vol. 8, no. 1, pp. 18–27, 2014.

[9] W. M. P. van der Aalst, N. Lohmann, P. Massuthe, C. Stahl, and K. Wolf, "From public views to private views—Correctness-by-design for services," in *Proc. 4th Int. Workshop Web Service Formal Methods*. Berlin, Germany: Springer, 2007, pp. 139–153.

[10] W. Aalst and M. Weske, *The P2P Approach to Interorganizational Workflows*. Berlin, Germany: Springer, 2013.

[11] M. Qi *et al.*, "An approach to extract public process from private process for building business collaboration," *J. Comput. Res. Develop.*, vol. 54, no. 9, pp. 1892–1908, 2017.

[12] J.-D. Ge, H.-Y. Hu, Y. Zhou, H. Hu, D.-Y. Wang, and X.-B. Guo, "A decomposition approach with invariant analysis for workflow coordination," *Chin. J. Comput.*, vol. 35, no. 10, pp. 2169–2181, 2012.

[13] W. M. P. van der Aalst, "The application of Petri nets to workflow management," *J. Circuits, Syst., Comput.*, vol. 8, no. 1, pp. 21–66, 1998.

[14] R. Eshuis, A. Norta, O. Kopp, and E. Pitkanen, "Service outsourcing with process views," *IEEE Trans. Services Comput.*, vol. 8, no. 1, pp. 136–154, Jan. 2015.

[15] D. Fei *et al.*, "Collaboration business process modeling based on Petri nets and Pi calculus," *J. Frontiers Comput. Sci., Technol.*, vol. 9, no. 6, pp. 692–706, 2015.

[16] 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, 2010.

[17] W. M. P. van der Aalst, "Workflow verification: Finding control-flow errors using Petri-net-based techniques," *Bus. Process Manage., Models, Techn., Empirical Stud.*, vol. 1806, no. 2, pp. 161–183, 2000.

[18] W. Fdhila, C. Indiono, S. Rinderle-Ma, and M. Reichert, "Dealing with change in process choreographies: Design and implementation of propagation algorithms," *Inf. Syst.*, vol. 49, pp. 1–24, Apr. 2015.

[19] Y. Min, H. Zhiqiu, and H. Jun, "Modeling and verification of cross-organizational multi-business transactions," *J. Softw.*, vol. 23, no. 3, pp. 517–538, 2012.

[20] Z. Guangquan, R. Mei, and W. Sheng, "Research on interaction modeling and mismatch checking of time-aware Web services," *Acta Electronica Sinica*, vol. 39, no. 11, pp. 2568–2575, 2011.

[21] Y. Chongyi, *The Principle and Application of Petri Nets*. Beijing, China: Electronic Industry Press, 2005.

[22] N. Lohmann and J. Kleine, "Fully-automatic translation of open workflow net models into simple abstract BPEL processes," *Lect. Notes Inform.*, vol. 4, no. 3, pp. 57–72, 2008.

[23] A. Polyvyanyy, L. Garcia-Banuelos, and M. Dumas, "Structuring acyclic process models," *Inf. Syst.*, vol. 37, no. 6, pp. 518–538, 2010.

[24] W. Song and H.-A. Jacobsen, "Static and dynamic process change," *IEEE Trans. Serv. Comput.*, vol. 11, no. 1, pp. 215–231, Jan./Feb. 2016.

[25] W. Aalst, J. Desel, and A. Oberweis, *Business Process Management: Models, Techniques, and Empirical Studies*. Berlin, Germany: Springer, 2000.

[26] T. H. Cormen, C. E. Leiserson, R. L. Rivest and C. Stein, *Introduction to Algorithms*, 3rd ed. Cambridge, MA, USA: MIT Press, 2013.

[27] M. Borkowski, W. Fdhila, M. Nardelli, C. Rinderle-Ma, and S. Schulte, "Event-based failure prediction in distributed business processes," *Inf. Syst.*, to be published, doi: 10.1016/j.is.2017.12.005.

[28] G. Matthias, P. Pascal, and S. Gwen, "VerChor: A framework for the design and verification of choreographies," *IEEE Trans. Services Comput.*, vol. 9, no. 4, pp. 647–660, Jul. 2016.

[29] A. Marco, I. Paola, and T. Massimo, "Choreography realizability enforcement through the automatic synthesis of distributed coordination delegates," *Sci. Comput. Program.*, vol. 160, no. 1, pp. 3–29, 2018.

[30] 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, 2011.

[31] C. Flavio, 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, no. 15, pp. 35–70, 2018.

[32] 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.

[33] 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, 2018.

[34] 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.

[35] P. Y. H. Wong and J. Gibbons, "Formalisations and applications of BPMN," *Sci. Comput. Program.*, vol. 76, no. 8, pp. 633–650, 2011.

[36] 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, 2012.

[37] 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, 2017.

[38] Q. Sheng, Z. Maamar, L. Yao, C. Szabo, and S. Bourne, "Behavior modeling and automated verification of Web services," *Inf. Sci.*, vol. 258, no. 3, pp. 416–433, 2014.

[39] C. Dechsupa, W. Vatanawood, and A. Thongtak, "Transformation of the BPMN design model into a colored Petri net using the partitioning approach," *IEEE Access*, vol. 6, pp. 38421–38436, 2018.

[40] W. Liu, P. Wang, Y. Du, M. Zhou, and C. Yan, "Extended logical Petri nets-based modeling and analysis of business processes," *IEEE Access*, vol. 5, pp. 16829–16839, 2017.

[41] R. Eshuis, A. Norta, and R. Roulaux, "Evolving process views," *Inf. Softw. Technol.*, vol. 80, pp. 20–35, Dec. 2016.

[42] W. Song, F. Chen, H.-A. 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.

QI MO received the B.S. degree from Huaibei Normal University, Huaibei, China, in 2009, and the M.S. and Ph.D. degrees in software engineering from Yunnan University, Kunming, China, in 2012 and 2015, respectively, where he is currently a Lecturer. 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.

FEI DAI received the B.S., M.S., and Ph.D. degrees 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 and software engineering.

DI LIU received the B.S. and M.S. degrees from Northwestern Polytechnical University, China, in 2007 and 2011, respectively, and the Ph.D. degree from Leiden University, The Netherlands, in 2017. He is currently an Assistant Professor with Yunnan University, China, and also holds a guest researcher position at Leiden University, The Netherlands. His research interests include real-time systems, energy-efficient multicore/manycore systems, and cyber-physical systems.

JIANGLONG QIN received the B.S., M.S., and Ph.D. degrees 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.

ZHONGWEN XIE received the B.S. degree from the Harbin Institute of Technology, Harbin, China, in 2009, and the M.S. and Ph.D. degrees 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. His research interests are in formal methods 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. His research interests include software process and software engineering.