IEEE Access
Multidisciplinary : Rapid Review : Open Access Journal

Received January 30, 2020, accepted February 16, 2020, date of publication March 16, 2020, date of current version March 26, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2980891

A Choreography Analysis Approach for Microservice Composition in Cyber-Physical-Social Systems

FEI DAI[1,3], QI MO[2,3], ZHENPING QIANG[1], BI HUANG[1], WEILI KOU[1], AND HONGJI YANG[4]

[1]School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming 650224, China
[2]School of Software, Yunnan University, Kunming 650091, China
[3]Key Laboratory of Software Engineering of Yunnan Province, Kunming 650091, China
[4]Department of Informatics, University of Leicester, Leicester LE1 7RH, U.K.

Corresponding author: Qi Mo (moqiyueyang@163.com)

This work was supported in part by the Project of National Natural Science Foundation of China under Grant 61702442 and Grant 61862065, and in part by the Application Basic Research Project in Yunnan Province under Grant 2018FB105.

ABSTRACT Choreography-driven microservice composition has provided a better way to integrate components in the Cyber-physical-Social System (CPSS). Choreography is a global contract that specifies interactions among microservices participating in a composite service. After modeling a choreography, a problem arises here is whether the choreography specification at design time can be implemented correctly by generated microservices that interact with each other via exchanging messages. In this paper, we propose a novel approach for choreography analysis. Specifically, a choreography is specified using a Labeled Transition Systems (LTSs); then, the microservices participating in a composite service can be generated from the given choreography via projection and ε-remove; finally, the analysis of the choreography can be checked for both synchronous and asynchronous compositions using refinement checking. Our approach is completely automated under the support of our developed tool and the Process Analysis Toolkit (PAT) tool.

INDEX TERMS Service composition, microservice, choreography, cyber-physical-social system, peer to peer communication.

I. INTRODUCTION

Cyber-physical-Social System (CPSS) comprises physical, cyber, and social worlds with various resources as services [1]–[4]. On one hand, CPSS emphasizes a huge number of deep interactions among the three worlds, so many approaches are proposed to manage these complex interactions at design time, such as planning based approaches, synthesis based approaches, and model-driven approaches [5]–[7]. On the other side, with the rapid development of edge computing [8], [13], [14], 5G [12], [15], [41], and mobile computing [16], [17], cloud applications have undergone a shift from monolithic applications to microservices [18]. Microservice architecture [18] is an architectural style that is a variant of service-oriented architecture (SOA) structural style, where microservices are fine-grained and lightweight. Duo to the characteristics of microservices, choreography-driven microservice composition approach is considered to provide a better way to integrate these CPSS services than traditional service composition approaches] [19]–[21].

It is well known that service composition is hot research in cloud computing paradigm. It not only considers the functional attributes of a composite service to match CPSS users' demands but also considers the nonfunctional attributes (e.g., Quality of Service) to satisfy CPSS users' requirements [9]–[11]. In this paper, we only focus on interactions among microservices from the functional attributes perspective.

Choreography is a global contract that each microservice should adhere to [22]. Service architect uses choreographies to model interactions among microservices from a global perspective when designing a composite service. After modeling a choreography, a problem arises here is whether the choreography specification at design time can be implemented correctly by a set of microservices from different words that interact with each other via exchanging messages. If we can

The associate editor coordinating the review of this manuscript and approving it for publication was Xiaokang Wang

find interaction faults from choreographies at design time, composition microservices may encounter a failure (e.g., deadlockor malfunction) during its execution. The process of finding interaction faults of choreographies is known as choreography analysis.

A large number of approaches have been proposed to analyze choreographies based on various formalisms such as automata-theory [26]–[29], process algebras [33], [34], and Petri nets [35], [36]. However, few approaches focus on analyzing choreographies under peer-to-peer communication. Besides, CPSSs bring two new challenges for microservice composition.

1) The complexity of choreography analysis grows rapidly with the number of CPSS services participating in a composite microservice. Usually, a composite service often consists of thousands of microservices [23].

2) In the context of microservice composition, the asynchronous communication model can either peer-to-peer communication or mailbox communication [24]. However, existing research mainly focuses on mailbox communication. Compared with mailbox communication, peer-to-peer communication is much more complex due to the increasing number of buffers.

To address the above two issues, we propose a choreography analysis approach for microservice composition in CPSSs. Our contribution can be summarized as follows.

1) Problem: we propose the problem of choreography analysis for microservice composition in CPSSs for the first time.

2) Approach: our approach considers choreography analysis for both synchronous and peer-to-peer composition for the first time.

3) Implementation: our approach can be completely automated by we developed a tool and the Process Analysis Toolkit (PAT) tool [25].

The rest of this paper is organized as follows. Section II introduces the problem definition and related work. Section III introduces the details of our approach. Section IV discusses the implementation of our approach and some experimental results. Section V concludes the paper.

II. PROBLEM DEFINITION AND RELATED WORK

A. PROBLEM DEFINITION

In the context of microservice composition, choreography analysis is to check whether this choreography can be implemented correctly by a set of microservices that are generated from this choreography such that the interactions of these generated microservices can exactly match this choreography specification. This problem can be formulated as follows,

$$MS_1||MS_2||\dots||MS_n \stackrel{?}{=} Chor \quad (1)$$

where *chor* denotes the choreography specification, MS_i denotes the generated *i-th* microservice, || denotes microservice composition under synchronous and asynchronous communication models, and = denotes equivalence.

If the equation is not satisfied, it means that the choreography specification cannot be implemented correctly. Thus, the service architect should not further advance the design process of microservice composition and should repair the choreography.

B. RELATED WORK

There have been earlier works on choreography analysis based on automata-theory. In [26], Bultan *et al.* proposed a framework to model interactions of e-service compositions, where the notion of conservations is used to describe message sequences. Under this framework, peers (components) that are using Büchi automaton can interact with each other asynchronously and maintain a buffer for storing incoming messages. In [27], Fu *et al.* proposed three realizability conditions and proved that conversation protocols based on Büchi automaton which satisfy these three properties are realizable. The realizability refers to check whether the synthesized peers from a conversation protocol from projection can produce the same set of conservations that exactly match this conversation protocol. In [28], Fu *et al.* proposed a technique called synchronizability analysis to analyze the interactions among web services. The synchronizability means that the conservation set among web services under synchronous communication remains the same under asynchronous communication. In [29], Bultan presented a tool for checking the realizability of collaboration diagrams under the support of the Web Service Analysis Tool (WSAT) [30]. The WAST can be used to check the three realizability conditions. In [31], Basu *et al.* proposed necessary and sufficient conditions for realizability of choreographies. These choreographies include web service choreographies, Singularity OS channel contracts and UML collaboration diagrams based on synchronizability analysis.

Some works used process algebras to analyze choreographies. In [22], Slaün *et al.* proposed an approach for analyzing the realizability of choreographies using process algebra encodings. This approach can be automated under the support of the Construction and Analysis of Distributed Processes (CADP) toolbox [33]. In [34], Poizat *et al.* transformed BPMN 2.0 choreographies into LOTOS NT process algebras and then checked the realizability of these choreographies using equivalence checking.

Other works used Petri nets to check the realizability of choreographies. In [35], Decker *et al.* proposed a Petri net extension called interaction Petri nets to check the realizability of choreographies. However, this work depends on the synchronous assumption, i.e., interaction Petri nets can only be used to check the realizability problem for synchronous communication. In [36], Hélouët *et al.* used stochastic time Petri nets STPNs to check the realizability of production systems' schedules. The realizability can further refined into two types: boolean realizability and probabilistic realizability.

To sum up, our work is different from the above works in two aspects. First, existing works focus on the problem of choreography analysis under the mailbox communications while our approach focuses on analyzing choreographies under the peer-to-peer communications. Second, our

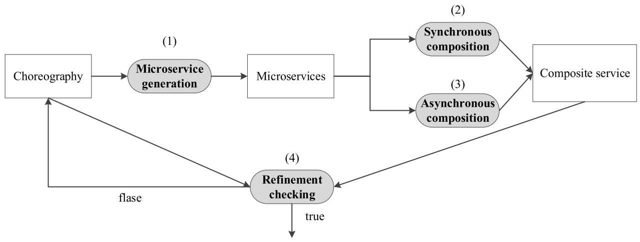


FIGURE 1. Overview of our approach.

approach is supported by tools for synchronous composition, asynchronous composition, and refinement checking in a completed automated way.

III. OUR PROPOSED APPROACH

A. OVERVIEW OF OUR APPROACH

In this section, our proposed approach is illustrated in Figure 1. The whole process of choreography analysis includes four steps:

Step 1: generate microservices from the given choreography.

Step 2: integrate these generated microservices under the synchronous composition.

Step 3: integrate these generated microservices under the asynchronous composition.

Step 4: compare the choreography specification with the composite service using refinement checking.

Labeled Transition Systems (LTSs) are used as the choreography specification language due to their simplicity and visualization.

Definition 1 (Choreography): A choreography $Chor = (S, M, \Delta, s_0, F)$ is a LTS, where

(1) S is a set of states.

(2) M is a message set.

(3) $\Delta \subseteq S \times M \times S$ is a transition relation.

(4) $s_0 \in S$ is the initial state.

(5) $F \subseteq S$ is a set of final states.

Definition 2 (Message Set): A message set M is a tuple (Σ, p, src, dst).

(1) Σ is a finite set of letters.

(2) $p \geq 1$ is a non-negative integer number which denotes the numbers of participating microservices.

(3) src and dst are functions that associate message $m \in \Sigma$ nonnegative integer numbers $src(m) \neq dst(m) \in \{1, 2, \ldots, p\}$.

We often write $m^{i \to j}$ for a message m such that $src(m) = i$ and $dst(m) = j$.

Figure 2 (a) shows a LTS choreography for online shopping that comes from in [web 06]. This is used as a running example throughout this paper. The initial states are subscripted with 0 and marked with incoming half-arrows. The final states are marked with double circles. The label of each transition is a message of the form $m^{i \to j}$, where i denotes the message sender and j denotes the message receiver.

This choreography shown in Figure 2 includes three microservices: Customer, Vendor, and Warehouse, that describes interactions as follows:

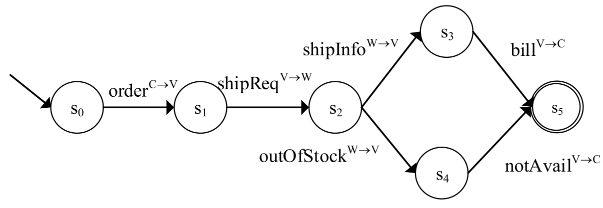


FIGURE 2. Choreography specification of online shopping.

The customer sends a ‘‘order message’’ to the Vendor and then the Vendor sends a shipReq message to the Warehouse. If the ordered item is in stock, the warehouse sends a ‘‘shipInfo message’’ to the vendor and then the vendor sends a ‘‘bill message’’ to the customer. If not, the warehouse sends an ‘‘out-of-stock message’’ to the vendor and then the vendor sends a ‘‘notAvailable message’’ to the customer.

B. MICROSERVICE GENERATION

Definition 3 (Microservice): A microservice $MS = (S, A, s_0, F, \delta)$ is a LTS, where:

(1) S is the finite set of states.

(2) A is a set of actions

(3) s_0 is the initial state.

(4) $F \subseteq S$ is the finite set of final states.

(5) $\delta \subseteq S \times (A \cup \{tau\}) \times S$ is the transition relation.

An action over M is either send message action $!m^{i \to j}$ or receive message action $?m^{i \to j}$, with $m \in$. A trace $\gamma \in M^*$ is a finite sequence of actions.

In a microservice, a transition $t \in \delta$ can be one of the following three types:

(1) a send message transition $(s_1, !m^{1 \to 2}, s_2)$ denotes that the microservice MS_1 sends a message m to another microservice MS_2 where $m \in$.

(2) a receive message transition $(s_1, ?m^{1 \to 2}, s_2)$ denotes that the microservice MS_1 consumes a message m from the microservice MS_2 where $m \in$.

(3) an ε-transition (s_1, ε, s_2) denotes that the invisible action of MS_1.

We often write $s_m \xrightarrow{!m^{1 \to 2}} s_k$ to denote that $(s_m, !m^{1 \to 2}, s_k)$.

Given a choreography, microservices participating in a composite service can be generated from the choreography via projection and ε -remove.

Definition 4 (Projection): The projection of a choreography $Chor = (S, M, \Delta, s_0, F)$ on one of the microservices MS_i is generated by performing the following operations.

(1) If a transition is $(s_m, m^{i \to j}, s_k)$ then replace it with $(s_m, !m^{i \to j}, s_k)$, i.e., this transition denotes microservices MS_i is the sender of message m.

(2) If a transition is $(s_m, m^{j \to i}, s_k)$ then replace it with $(s_i, ?m^{j \to i}, s_j)$, i.e., this transition denotes microservices MS_i is the receiver of message m.

(3) Otherwise, replace the transition $(s_m, m^{x \to y}, s_k)$ with (s_m, ε, s_k).

For the resulting $Chor$, we can remove ε transitions and determines the $Chor$ to obtain the generated microservice [27], [31].

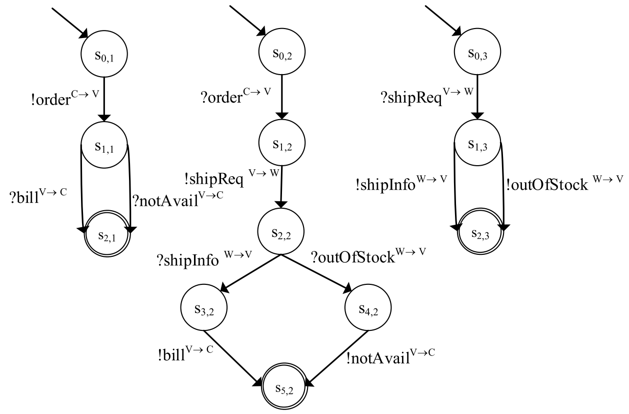


FIGURE 3. Microservices generated from the choreography shown in Figure 1: (a) customer, (b)vendor, and (c) warehouse.

Figure 3 shows three microservices that are generated from our running example.

Once microservices are generated, it's very difficult to judge the interactions among these microservices are the same as the given choreography specification. In the following sections, we answer this question using refinement checking.

C. SYNCHRONOUS COMPOSITION

In synchronous communication, every send message action is consumed followed by a receive message action [31], i.e., the send messages action and the corresponding receive message actions are executed simultaneously.

Below we define the synchronous composition through synchronous communication.

Definition 5 (Synchronous Composition): Given a set of microservices $MSs = (MS_1, MS_2, \ldots, MS_n)$ where $MS_i = (S_i, A_i, \delta_i, s_{0i}, F_i,)$ and A_i over $M_i = (\Sigma_i, p_i, src_i, dst_i)$, the synchronous composition $(MS_1||_s MS_2||_s \ldots ||_s MS_n)$ is a labeled transition system $I_s = (C, M, c_0, \Delta,)$ where:

- $C \subseteq S_1 \times S_2 \ldots \times S_n$ is the set of states.
- $M = \cup_i M_i$ is the set of messages.
- $c_0 \in C$ is the initial state.
- $\Delta \subseteq C \times (M \cup \{\varepsilon\}) \times C$ for $c = (s_1, s_2, \ldots, s_n)$ and $c' = (s_1', s_2', \ldots, s_n')$.

(a) $c \xrightarrow{m^{i\to j}} c' \in \Delta$ *if* $\exists i, j \in \{1, 2, \ldots, n\} \wedge m \in M$:
(i) $src(m) = i \wedge dst(m) = j$,
(ii) $s_i \xrightarrow{!m^{i\to j}} s_i' \in \delta_i$,
(iii) $s_j \xrightarrow{?m^{i\to j}} s_j' \in \delta_j$,
(iv) $\forall k \in \{1, 2, \ldots, n\} : k \neq i \wedge k \neq j \Rightarrow s_k' = s_k$.
[synchronous send-receive message action]
(b) $c \xrightarrow{\varepsilon} c' \in \Delta$ *if* $\exists i, j \in \{1, 2, \ldots, n\}$:
(i) $s_i \xrightarrow{\varepsilon} s_i' \in \delta_i$,
(ii) $\forall k \in \{1, 2, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$.
[internal action]

D. ASYNCHRONOUS COMPOSITION

In asynchronous communication, microservices interact with each other asynchronously through unbound buffers.

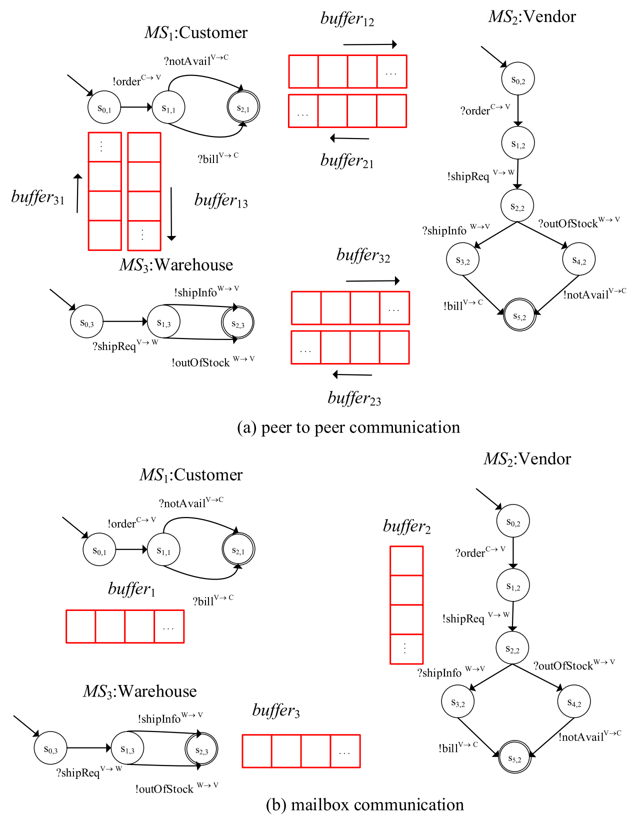


FIGURE 4. Peer-to-peer communication vs mailbox communication.

A microservice can either send messages to the buffers of other microservices or receive messages from its buffers.

There are two different semantics for asynchronous communication: peer-to-peer communication and mailbox communication. The mailbox communication shown in Figure 4(b) requires all messages sent to MS_1 from the other microservices are stored in a buffer (i.e., a message queue) that is specific to MS_1. The peer-to-peer communication shown in Figure 4 (a) requires each message sent from a microservice MS_1 to another microservice MS_2 is stored in a buffer in a FIFO fashion which is specific to the pair (MS_1, MS_2). In this paper, we focus on peer-to-peer communication.

In the peer-to-peer semantics, each participating microservice of a composite service is equipped with buffers for different incoming messages from other microservices [32]. A microservice MS_1 either sends a message to the buffer $buffer_{1j}$ of another microservice MS_j, or consume a message from its buffers $Q_1 = (buffer_{j1})$ where $j \neq 1$, or perform an internal action.

Below we define the asynchronous composition through peer-to-peer communication.

Definition 6 (Asynchronous Composition): Given a set of microservices $MSs = (MS_1, MS_2, \ldots, MS_n)$ where $MS_i = (S_i, A_i, \delta_i, s_{0i}, F_i,)$ and Q_i being its buffers, the asynchronous composition $(MS_1||_a MS_2 ||_a \ldots ||_a MS_n)$ is a labeled transition system $I_a = (C, M, c_0, \Delta)$ where:

- $C \subseteq Q_1 \times S_1 \times Q_2 \times S_2 \ldots Q_n \times S_n$ is the set of states such that $\forall i \in \{1, 2, \ldots, n\} : Q_i = (buffer_{ji})$, where $\forall j \in \{1, 2, \ldots, n\} \wedge i \neq j \wedge buffer_{ji} \subseteq (M_j)^*$.
- $M = \cup_i M_i$ is the set of messages.
- $c_0 \in C$ is the initial state such that $c_0 = (\underbrace{(\{\}, \{\}, \ldots \{\})}_{n-1}, c_{01}, \underbrace{(\{\}, \{\}, \ldots \{\})}_{n-1}, c_{02}, \ldots, \underbrace{(\{\}, \{\}, \ldots \{\})}_{n-1}, c_{0n})$ where $c_{0i} = MS_i.\ s_{0i}$.
- $\Delta \subseteq C \times (M \cup \{\varepsilon\}) \times C$ for $c = (Q_1, s_1, Q_2, s_2, \ldots, Q_n, s_n)$ and $c' = (Q'_1, s'_1, Q'_2, s'_2, \ldots, Q'_n, s'_n)$

(a) $c \xrightarrow{!m^{i \to j}} c' \in \Delta$ *if* $\exists i, j \in \{1, 2, \ldots, n\} \wedge m \in M$:
(i) $src(m) = i \wedge dst(m) = j$,
(ii) $s_i \xrightarrow{!m^{i \to j}} s'_i \in \delta_i$,
(iii) $\forall k \in \{1, 2, \ldots, n\} : k = i \Rightarrow buffer'_{kj} = buffer_{kj}m$,
(iv) $\forall k \in \{1, 2, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$,
(v) $\forall k, l \in \{1, 2, \ldots, n\} : k \neq i \wedge l \neq j \wedge k \neq l \Rightarrow buffer'_{kl} = buffer_{kl}$.
[send message action]

(b) $c \xrightarrow{?m^{i \to j}} c' \in \Delta$ *if* $\exists i, j \in \{1, 2, \ldots, n\} \wedge m \in M$:
(i) $src(m) = i \wedge dst(m) = j$,
(ii) $s_j \xrightarrow{?m^{i \to j}} s'_j \in \delta_j$,
(iii) $\forall k \in \{1, 2, \ldots, n\} : k = i \Rightarrow buffer'_{kj} = mbuffer_{kj}$,
(iv) $\forall k \in \{1, 2, \ldots, n\} : k \neq j \Rightarrow s'_k = s_k$,
(v) $\forall k, l \in \{1, 2, \ldots, n\} : k \neq i \wedge l \neq j \wedge k \neq l \Rightarrow buffer'_{kl} = buffer_{kl}$.
[receive message action]

(c) $c \xrightarrow{\varepsilon} c' \in \Delta$ *if* $\exists i, j \in \{1, 2, \ldots, n\}$:
(i) $s_i \xrightarrow{\varepsilon} s'_i \in \delta_i$,
(ii) $\forall k \in \{1, 2, \ldots, n\} : k \neq i \Rightarrow s'_k = s_k$,
(iii) $\forall k \in \{1, 2, \ldots, n\} : Q'_k = Q_k$.
[internal action]

According to Definition 6, there are three following interaction types in a composite service under the peer-to-peer semantics.

(1) a send message action $c \xrightarrow{!m^{i \to j}} c'$ denotes that microservice MS_i sends a message m to another microservice MS_j where $m \in M_i$ (6-i). After that, the state of the sender is changed (6a-ii), the message will be inserted to the tail of the $buffer_j$ of the receiver (6a-iii), the other microservices' states do not change (6a-iv), and the other buffers do not change (6a-v).

(2) a receive message action $c \xrightarrow{?m^{i \to j}} c'$ denotes that microservice MS_j consumes a message m sent from microservice MS_i where $m \in M_i$ (6b-i). After that, the state of the receiver is changed (6b-ii), the message at the head of the $buffer_j$ of the receiver will be consumed (9b-iii), the other microservices' states do not change (9b-iv), and the other buffers do not change (9b-v).

(3) an internal action $c \xrightarrow{\varepsilon} c'$ denotes that microservice MS_i executes an internal action (6c-i). After that, the other microservices' states do not change (6c-ii) and the other buffers also do not change (6c-iii).

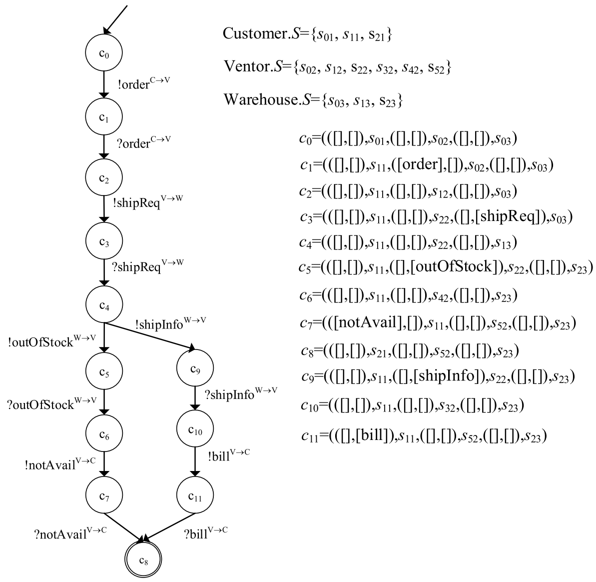


FIGURE 5. The 1-bounded asynchronous composite service.

In the asynchronous composition, the interactions among microservices depend on the order the send and receive actions as well as the size of the buffers associated with each microservice [29]. When buffers are unbounded, the interactions may be infinite.

We define the bounded asynchronous composition in the following.

Definition 7 (k-Bounded Asynchronous Composition): Given a set of microservices $MSs = (MS_1, MS_2, \ldots, MS_n)$ where $MS_i = (S_i, A_i, \delta_i, s_{0i}, F_i,)$ and Q_i being its buffers of size k, the k-bounded asynchronous composition $(MS_1 ||_a^k MS_2 ||_a^k \ldots ||_a^k MS_n)$ is a labeled transition system $I_a^k = (C, M, c_0, \Delta)$ and described by augmenting condition 6(a) in Definition 6 to include the condition $Q_j = (q_1, q_{j-1}, q_{j+1}, \ldots, q_n)$: $|q_j| < \text{k}$, where $|q_i|$ denotes the length of the buffers for microservice MS_i.

In the k-bounded asynchronous composition, the send message actions are blocked if the receiver's buffer contains k messages. Compared with the unbounded asynchronous composite service, the interactions of a k-bounded asynchronous composite service are finite.

For our running example, Figure 5 shows the 1-bounded asynchronous composite service.

E. REFINEMENT CHECKING

In this section, we analyze choreographies by comparing the choreography specification with the composite service composed of interacting microservices using refinement checking. A choreography specification can be implemented correctly if the interaction traces of the composite service are the same as in the given choreography.

Therefore, analyzing choreography includes three steps:

1) generate all the interaction traces of the choreography *Chor*.

2) generate all the interaction traces of a composite service I under synchronous or asynchronous composition.

TABLE 1. Analysis results for some cases.

id	description	*Chor*		I_s			Choreography analysis		
		(*states/transitions*)	|*MSs*|	(*states/transitions*)	I_a^1 (*stats/transitions*)	I_a^2 (*stats/transitions*)	syn	async[1]	async[2]
case-2	booking system[34]	8/8	4	10/11	23/29	24/31	×	×	×
ase-15	train station[22]	9/10	4	11/13	44/76	64/122	√	×	×
case-18	Protocol[27]	6/7	3	7/14	26/62	35/91	√	×	×
case-25	online shopping[38]	6/6	3	7/7	13/13	13/13	√	√	√
case-27	cloud application[39]	4/5	4	6/10	96/240	too large	×	×	×
case-34	figure 2[40]	3/3	3	3/3	10/14	15/24	√	√	√
case-38	figure 1[35]	32/31	4	11/14	28/43	28/43	√	√	√
case-40	composition 2[28]	3/5	2	5/6	10/11	10/11	√	√	√
case-42	Figure 8[29]	7/7	3	7/6	16/20	16/20	√	×	×
case-45	Figure 1[32]	7/6	3	6/5	13/15	20/26	×	×	×

3) refinement checking between *traces*(*Chor*) and *traces*(*I*).

Definition 8 (Trace Refinement): Let $I = (C, M, c_{0,} \Delta)$ be a LTS for a choreography implementation, i.e., a composite service, and $Chor = (S, M, \Delta, s_0, F)$ be a LTS for a choreography specification, *I refines Chor* if $traces(I) \subseteq traces(Chor)$.

Based on the refinement relations, the analysis of the choreography can be checked for both synchronous and bounded asynchronous compositions.

A choreography $Chor = (S, M, \Delta, s_0, F)$ is implemented correctly under synchronous composition iff I_s *refines Chor* and *Chor refines*(I_s), i.e., $traces(I_s) = traces(Chor)$, where $I_s = (MS_1, MS_2, \ldots, MS_n)$ and all microservices are generated from the Chor.

A choreography $Chor = (S, M, \Delta, s_0, F)$ is implemented correctly under asynchronous composition iff I_a^k *refines Chor* and *Chor refines* I_a^k, where $I_a^k = MS_1 ||_a^k MS_2 ||_a^k \ldots ||_a^k MS_n$ and all microservices are generated from the *Chor* and are equipped with buffers of size *k*.

Note that during the process of refinement checking, we only consider the send message actions, ignoring the receive message actions, because the receive message actions refer to local consumptions by microservices from their buffers [37] and can be considered as invisible actions.

IV. TOOL SUPPORT AND EXPERIMENTS

The four steps of our approach are completely automated. For step 1 in Fig.1, we have developed a tool named **chor2ms** to generates the participating microservices from a given choreography. For steps 2 and 3 in Fig.1, The synchronous and asynchronous compositions are achieved using the PAT simulator in the PAT tool. For step 4 in Fig.1, the refinement checking is achieved using the PAT verifier in the PAT tool.

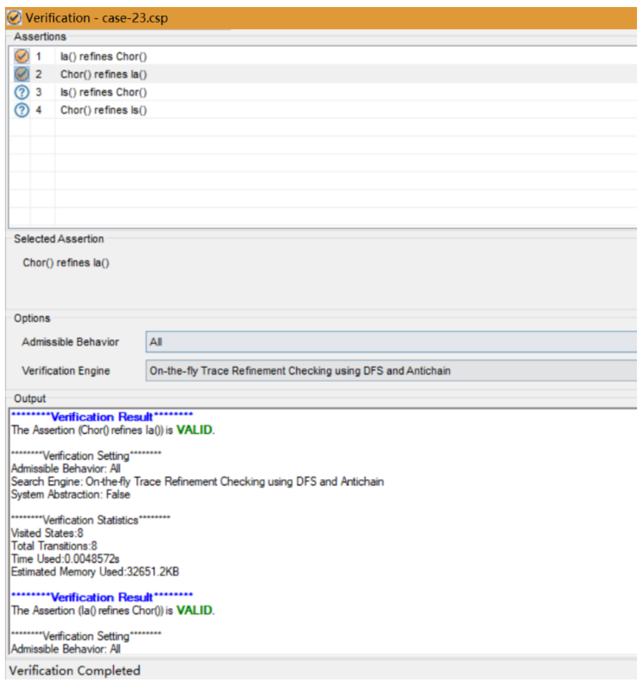


FIGURE 6. The screenshot of the analysis results.

The screenshot of the analysis results is shown in Fig. 6, where the running example in this paper is checked.

Our approach was validated on 50 cases obtained from research papers. All cases were carried out on a PC with 2.50GHz Processor and 8GB of RAM, running Windows 10.

Table 1 shows some experimental results. For each case, the table gives the choreography's description (*description*), the choreography's size (*Chor*), and the number participating microservices (*MSs*). Next, the table gives the synchronous composite service's size (I_s), the 1-bounded asynchronous composite service's size (I_a^1), and the 2-bounded asynchronous composite service's size (I_a^2). Last, the table gives analysis results for both synchronous and bounded asynchronous compositions, where "×" denotes that the choreography cannot be implemented correctly sound and "+" denotes that the choreography can be implemented correctly.

During the experiments, the case-27 faces the state space explosion problem. In case-27, the state space of 2-bounded asynchronous composition increases exponentially with the size of buffers. "Too large" means that the PAT simulator is forced to stop due to the huge state space size (>300 states)

Out of the 10 cases presented in Table 1, 6 choreographies (case-2, case -15, case-18, case-27, case-42 and case 45) cannot be implemented correctly. This means that designing choreography for microservice composition is error-prone, especially in the context of CPSS.

V. CONCLUSION

In this paper, we have investigated the problem of choreography analysis. Analyzing refers to check whether a choreography can be implemented correctly by a set of microservices generated from the given choreography via projection and ε -remove. In our approach, LTSs are used as the choreography specification language. Our approach can analyze choreographies under synchronous and asynchronous compositions using refinement checking and be automated by the use of the tool we developed and the PAT tool.

The future work is to investigate the relationships between finite interaction sequences and unbound buffers in the case of asynchronous composition.

REFERENCES

[1] S. Wang, A. Zhou, M. Yang, L. Sun, C. Hsu, and F. Lee, "Service composition in cyber-physical-social systems," *IEEE Trans. Emerg. Topics Comput.*, vol. 8, no. 1, pp. 82–91, Jan./Mar. 2020.

[2] H. Song, R. Srinivasan, T. Sookoor, and S. Jeschke, *Smart Cities: Foundations, Principles, and Applications*. Hoboken, NJ, USA: Wiley, 2017, pp. 1–906.

[3] H. Song, S. Jeschke, and G. A. Fink, *Security and Privacy in Cyber-Physical Systems: Foundations, Principles and Applications*. Chichester, U.K.: Wiley, 2017, pp. 1–472.

[4] L. Qi, Y. Chen, Y. Yuan, S. Fu, X. Zhang, and X. Xu, "A QoS-aware virtual machine scheduling method for energy conservation in cloud-based cyber-physical systems," *World Wide Web J.*, vol. 23, no. 2, pp. 1275–1297, Mar. 2020, doi: 10.1007/s11280-019-00684-y.

[5] K. Sangsanit, W. Kurutach, and S. Phoomvuthisarn, "REST Web service composition: A survey of automation and techniques," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Chiang Mai, Thailand, 2018, pp. 116–121.

[6] S. Dustdar and W. Schreiner, "A survey on Web services composition," *Int. J. Web Grid Services*, vol. 1, no. 1, pp. 1–30, 2005.

[7] Y. Jiang, H. Song, Y. Yang, H. Liu, M. Gu, Y. Guan, J. Sun, and L. Sha, "Dependable model-driven development of CPS: From stateflow simulation to verified implementation," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 1, 2019, Art. no. 12.

[8] X. Xu, R. Mo, F. Dai, W. Lin, S. Wan, and W. Dou, "Dynamic resource provisioning with fault tolerance for data-intensive meteorological workflows in cloud," *IEEE Trans Ind. Informat.*, to be published, doi: 10.1109/TII.2019.2959258.

[9] H. Liu, H. Kou, C. Yan, and L. Qi, "Link prediction in paper citation network to construct paper correlation graph," *EURASIP J. Wireless Commun. Netw.*, vol. 2019, no. 1, pp. 1–12, Dec. 2019, doi: 10.1186/s13638-019-1561-7.

[10] W. Gong, L. Qi, and Y. Xu, "Privacy-aware multidimensional mobile service quality prediction and recommendation in distributed fog environment," *Wireless Commun. Mobile Comput.*, vol. 2018, pp. 1–8, Apr. 2018.

[11] L. Qi, X. Zhang, W. Dou, C. Hu, C. Yang, and J. Chen, "A two-stage locality-sensitive hashing based approach for privacy-preserving mobile service recommendation in cross-platform edge environment," *Future Gener. Comput. Syst.*, vol. 88, pp. 636–643, Nov. 2018.

[12] M. A. Rahman, M. M. Hasan, A. T. Asyhari, and M. Z. A. Bhuiyan, "A3D-collaborative wireless network: Towards resilient communication for rescuing flood victims," in *Proc. IEEE 15th Int. Conf Dependable, Auton. Secure Comput., 15th Int. Conf Pervas. Intell. Comput., 3rd Int. Conf Big Data Intell. Comput. Cyber Sci. Technol. Congr. (DASC/PiCom/DataCom/CyberSciTech)*, Nov. 2017, pp. 385–390.

[13] M. A. Rahman, M. Y. Mukta, A. Yousuf, A. T. Asyhari, M. Z. A. Bhuiyan, and C. Y. Yaakub, "IoT based hybrid green energy driven highway lighting system," in *Proc. IEEE Int. Conf. Dependable, Auton. Secure Comput., Int. Conf. Pervas. Intell. Comput., Int. Conf. Cloud Big Data Comput., Int. Conf. Cyber Sci. Technol. Congr. (DASC/PiCom/CBDCom/CyberSciTech)*, Aug. 2019, pp. 587–594.

[14] W. Wei, M. A. Rahman, I. F. Kurniawan, A. T. Asyhari, S. M. N. Sadat, and L. Yao, "Immune genetic algorithm optimization and integration of logistics network terminal resources," in *Proc. 3rd IEEE Int. Conf. Robot. Comput. (IRC)*, Naples, Italy, Feb. 2019, pp. 435–436.

[15] X. Xu, Y. Xue, L. Qi, Y. Yuan, X. Zhang, T. Umer, and S. Wan, "An edge computing-enabled computation offloading method with privacy preservation for Internet of connected vehicles," *Future Gener. Comput. Syst.*, vol. 96, pp. 89–100, Jul. 2019.

[16] X. Xu, C. He, Z. Xu, L. Qi, S. Wan, and M. Z. A. Bhuiyan, "Joint optimization of offloading utility and privacy for edge computing enabled IoT," *IEEE Internet Things J.*, to be published, doi: 10.1109/JIOT.2019.2944007.

[17] X. Xu, Q. Liu, X. Zhang, J. Zhang, L. Qi, and W. Dou, "A blockchain-powered crowdsourcing method with privacy preservation in mobile environment," *IEEE Trans. Comput. Social Syst.*, vol. 6, no. 6, pp. 1407–1419, Dec. 2019.

[18] A. Balalaie, A. Heydarnoori, and P. Jamshidi, "Microservices architecture enables DevOps: Migration to a cloud-native architecture," *IEEE Softw.*, vol. 33, no. 3, pp. 42–52, May 2016.

[19] X. Wang, L. T. Yang, X. Xie, J. Jin, and M. J. Deen, "A cloud-edge computing framework for cyber-physical-social services," *IEEE Commun. Mag.*, vol. 55, no. 11, pp. 80–85, Nov. 2017.

[20] X. Wang, W. Wang, L. T. Yang, S. Liao, D. Yin, and M. J. Deen, "A distributed HOSVD method with its incremental computation for big data in cyber-physical-social systems," *IEEE Trans. Comput. Social Syst.*, vol. 5, no. 2, pp. 481–492, Jun. 2018.

[21] X. Wang, L. T. Yang, Y. Wang, X. Liu, Q. Zhang, and M. J. Deen, "A distributed tensor-train decomposition method for cyber-physical-social services," *ACM Trans. Cyber-Phys. Syst.*, vol. 3, no. 4, pp. 1–15, Oct. 2019, doi: 10.1145/3323926.

[22] G. Salaün, T. Bultan, and N. Roohi, "Realizability of choreographies using process algebra encodings," *IEEE Trans. Services Comput.*, vol. 5, no. 3, pp. 90–302, Feb. 2012.

[23] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Delta debugging microservice systems with parallel optimization," *IEEE Trans. Services Comput.*, to be published.

[24] X. Zhou, X. Peng, T. Xie, J. Sun, C. Ji, W. Li, and D. Ding, "Fault analysis and debugging of microservice systems: Industrial survey, benchmark system, and empirical study," *IEEE Trans. Softw. Eng.*, to be published.

[25] J. Sun, Y. Liu, and J. Dong, "Model checking CSP revisited: Introducing a process analysis toolkit," in *Proc. Int. Symp. Leveraging Appl. Formal Methods, Verification Validation*, 2008, pp. 307–322.

[26] T. Bultan, X. Fu, R. Hull, and J. Su, "Conversation specification: A new approach to design and analysis of E-service composition," in *Proc. WWW*, Budapest, Hungary, May 2003, pp. 403–410.

[27] X. Fu, T. Bultan, and J. Su, "Conversation protocols: A formalism for specification and verification of reactive electronic services," *Theor. Comput. Sci.*, vol. 328, nos. 1–2, pp. 19–37, Nov. 2004.

[28] X. Fu, T. Bultan, and J. Su, "Synchronizability of conversations among Web services," *IEEE Trans. Softw. Eng.*, vol. 31, no. 12, pp. 1042–1055, Dec. 2005.

[29] T. Bultan, C. Ferguson, and X. Fu, "A tool for choreography analysis using collaboration diagrams," in *Proc. IEEE Int. Conf. Web Services*, Jul. 2009, pp. 856–863.

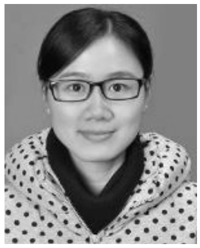
[30] X. Fu, T. Bultan, and J. Su, "WSAT: A tool for formal analysis of Web services," in *Proc. Int. Conf. Comput. Aided Verification*, in Lecture Notes in Computer Science, 2004, pp. 510–514.
[31] S. Basu, T. Bultan, and M. Ouederni, "Deciding choreography realizability," *ACM SIGPLAN Notices*, vol. 47, no. 1, pp. 191–202, Jan. 2012.
[32] A. Finkel and É. Lozes, "Synchronizability of communicating finite state machines is not decidable," in *Proc. 44th Int. Colloq. Automat., Lang., Program. (ICALP)*, vol. 122, 2017, pp. 1–14.
[33] H. Garavel, R. Mateescu, F. Lang, and W. Serwe, "CADP 2006: A toolbox for the construction and analysis of distributed processes," in *Proc. 19th Int. Conf. Comput. Aided Verification (CAV)*, 2007, pp. 158–163.
[34] P. Poizat and G. Salaün, "Checking the realizability of BPMN 2.0 choreographies," in *Proc. 27th Annu. ACM Symp. Appl. Comput. (SAC)*, 2012, pp. 1927–1934.
[35] G. Decker and M. Weske, "Local enforceability in interaction Petri nets," in *Proc. 5th Int. Conf. Bus. Process. Manage.*, 2007, pp. 305–319.
[36] L Hélouët and K. Kecir, "Realizability of schedules by stochastic time Petri nets with blocking semantics," *Sci. Comput. Program.*, vol. 157, pp. 71–102, Jun. 2018.
[37] L. Akroun, G. Salaün, and L. Ye, "Automated analysis of asynchronously communicating systems," in *Proc. 23rd Int. Symp. Model Checking Softw.*, 2016, pp. 1–18.
[38] T. Bultan, "Modeling interactions of Web software," in *Proc. 2nd Int. Workshop Automat. Specification Verification Web Syst. (WWV)*, Nov. 2006, pp. 45–52.
[39] M. Güdemann, G. Salaün, and M. Ouederni, "Counterexample guided synthesis of monitors for realizability enforcement," in *Proc. Int. Symp. Automat. Technol. Verification Anal.*, in Lecture Notes in Computer Science, vol. 7561, 2012, pp. 238–253.
[40] M. Ouederni, G. Salaün, and T. Bultan, "Compatibility checking for asynchronously communicating software," in *Formal Aspects of Component Software FACS* (Lecture Notes in Computer Science), vol. 8348, J. Fiadeiro, Z. Liu, and J. Xue, Eds. Cham, Switzerland: Springer, 2014.
[41] X. Wang, L. T. Yang, Y. Wang, L. Ren, and M. J. Deen, "ADTT: A highly-efficient distributed tensor-train decomposition method for IoT big data," *IEEE Trans Ind. Informat.*, to be published, doi: 10.1109/TII.2020.2967768.

FEI DAI received the Ph.D. degree from Yunnan University, Kunming, China, in 2011.

He is currently a Professor and a Full Professor with School of Big Data and Intelligent Engineering, Southwest Forestry University, Kunming. He has authored or coauthored more than 80 peer-reviewed research articles, including the IEEE TRANSACTIONS ON INDUSTRIAL INFORMATICS, the IEEE INTERNET OF THINGS JOURNAL, the IEEE TRANSACTIONS ON SERVICES COMPUTING, the IEEE TRANSACTIONS ON EMERGING TOPICS IN COMPUTATIONAL INTELLIGENCE, COMPLEXITY, and IEEE ACCESS. His research interests include service computing, edge computing, business process management (BPM), and software engineering.

QI MO received the Ph.D. degree from Yunnan University, Kunming, China, in 2015. He is currently a Lecturer with the School of Software, Yunnan University, China. His research interests are in formal methods and business process management (BPM).

ZHENPING QIANG received the Ph.D. degree from Yunnan University, Kunming, China, in 2018. He is currently an Associate Professor with the School of Big Data and Intelligent Engineering, Southwest Forestry University, China. His research interests are in machine learning, artificial intelligence, and big data.



BI HUANG received the M.S. degree in software engineering from Yunnan University, Kunming, in 2013. She is currently a Lecturer with the School of Big Data and Intelligent Engineering, Southwest Forestry University, China. Her research interests are in machine learning, artificial intelligence, and big data.

WEILI KOU received the Ph.D. degree from the Kunming University of Science and Technology, Kunming, China, in 2015. He is currently a Professor with the College of Big Data and Intelligent Engineering, Southwest Forestry University, China. His research interests are in image intelligent processing and analysis and temporal spatial big data.

HONGJI YANG received the B.S and M.S. degrees in computer science from Jilin University, China, in 1982 and 1985, respectively, and the Ph.D. degree in computer science from Durham University, U.K., in 1994. He is currently a Professor with the University of Leicester, U.K. He has published well over 400 refereed journal articles and conference papers. His research interests include creative computing, software engineering, and internet computing. In 2010, he became a member of the IEEE Computer Society Golden Core. He is the Editor-in-Chief of the *International Journal of Creative Computing*.