IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 13, 2020, accepted September 10, 2020, date of publication September 18, 2020,
date of current version September 29, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3024689

Detecting Data Flow Errors Across Processes in Business Process Collaboration

TIANHONG XIONG, MAOLIN PAN, YANG YU, AND DINGJUN LOU
School of Data and Computer Science, Sun Yat-sen University, Guangzhou 510006, China

Corresponding author: Yang Yu (yuy@mail.sysu.edu.cn)

This work was supported in part by the National Key Research and Development Program of China under Grant 2017YFB0202201, in part by the National Natural Science Foundation of China (NSFC) under Grant 61972427, and in part by the NSFC-Guangdong Joint Fund Project under Grant U1911205.

ABSTRACT In business process collaboration (BPC), especially when it comes to message communication and data exchange, there are complex data dependencies among sender process, receiver process and messages. However, each participant of the overall BPC develops its part independently as a service, including its own communication part and data flow. As a result, data flow errors across processes may occur easily. In this article, we propose a method based on BPMN to detect these errors caused by data dependency violations. Our method is inspired by the study of detecting data flow errors within a single process and focuses on a subset of the elements of the BPC model, without having to consider the complete set. In particular, we define a set of data flow error patterns by analyzing and formalizing data dependencies in order to clearly clarify and identify errors. Then we give the corresponding automatic detection algorithm. Finally, through two evaluations, we demonstrate the effectiveness of our proposal.

INDEX TERMS Business process management, business process collaboration, data flow error across processes, data dependency.

I. INTRODUCTION

With the development of business globalization, business process collaboration (BPC) among organizations has become more and more complex and frequent [1]. Meanwhile, the booming of BPaaS (Business Process as a Service) has facilitated the development and deployment of such service-based BPC [2]. In this context, it is important to ensure and maintain the soundness of BPC involving multiple participants.

In general, the soundness of data flow, like that of control flow, is a fundamental requirement of BPC [3]. Besides the soundness of data flow within each private (local) process, the BPC requires that the data flow across processes is sound. In fact, BPC enables multiple processes to cooperate and interact with each other to achieve shared goals [4]. To ensure successful collaboration, clear message communication and data exchange are required between processes. Thus, an error-free data flow across processes plays a crucial role. However, since each participant in BPC independently designs its own communication part and data flow, data flow errors across processes caused by data dependency violations are easy to occur.

To address the above issue, several methods have been proposed to consider data flows across processes in BPC [4]–[7]. However, less attention has been paid to providing the formalization of data flow errors across processes, which affects the identification and detection of errors. Indeed, when detecting data flow errors, semi-formal definitions, user-provided definitions, and natural text descriptions sometimes contain misleading information, which may become more serious in BPC context because different participants might have different understandings of errors. In addition, although some studies have proposed corresponding formalizations to clarify and detect data flow errors within a single process (e.g., Missing Data, Redundant Data, Lost Data [8]–[10]), they do not consider the data related to messages, and cannot support the message communication and data exchange across processes that BPC needs to cover.

In this article, we propose a formal method based on BPMN to detect data flow errors across processes in BPC. Our method draws on the ideas from both detecting data flow errors within a single process [8] and TraDE [11], [12]. The former describes the data dependencies among nodes within

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

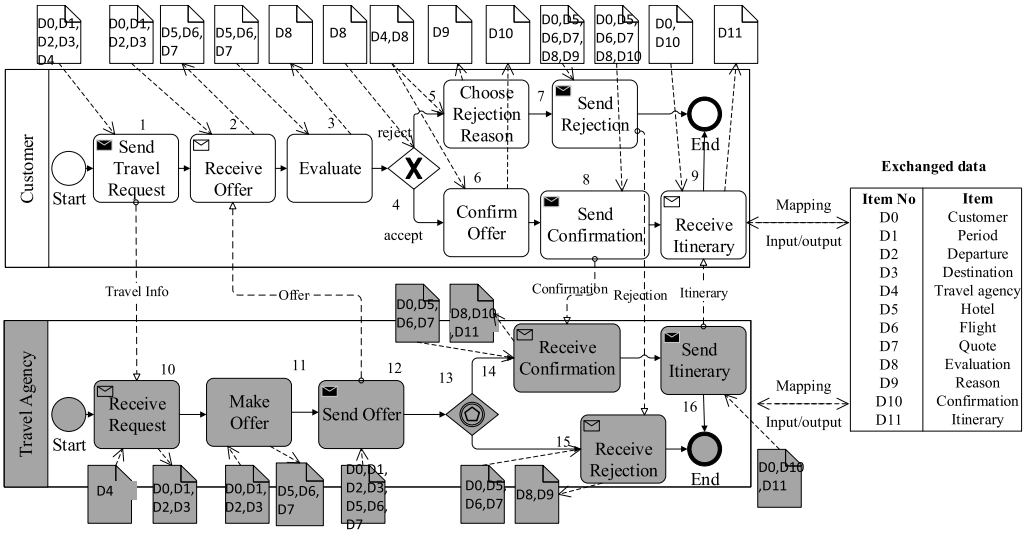

FIGURE 1. The collaboration model of the Booking Travel example. Each participant in BPC independently designs its own communication part and data flow (the Customer and Travel Agency are invisible to each other). Meanwhile, the exchanged data (right side) among the sender, receiver and messages (left side) is manually added and maintained by participants. In this context, the definition, use, and delivery of exchanged data is an error-prone task. Thus, it is necessary to provide a unified formalization for data flow errors across processes and accurately detect them.

a single process based on operational data specification (read and write operations), which inspires us to further analyze data dependencies across processes and formalize data flow error patterns across processes. The latter provides a data modeling way that unifies the data flow across and within processes, which makes the exchanged data open in the form of data contract agreed by all participants and easy to share and access in BPC. With these in mind, the key to detect data flow errors across processes is to check whether the exchanged data is accurately defined, used and passed in processes and messages, so as to ensure that the data dependencies among the sender, receiver and messages are not violated.

In summary, we make three primary contributions:

(1) We propose a formal method to detect data flow errors across processes, which is mainly applicable to message communication and data exchange in BPC. It focuses on a subset of the elements of the BPC model (e.g., sending/receiving nodes, messages, and exchanged data), without having to consider the complete set.

(2) We define a set of data flow error patterns across processes and a corresponding automatic detection algorithm. Especially in BPC context, unified and formal definitions can not only clearly clarify and identify errors, but also avoid deviations in participants' understanding of data flow errors.

(3) We conduct two evaluations with two collaboration model sets (8 and 33 models, respectively) to illustrate the effectiveness of our proposal, including a benchmark evaluation and a comparative evaluation.

The rest of this article is organized as follows. Section 2 provides an example throughout the paper to illustrate our motivations. Section 3 presents the definitions related to our method. Section 4 analyzes the data dependencies in BPC and formalizes them. Section 5 proposes the definitions of data flow error patterns and corresponding detection algorithm. After two evaluations are reported in Section 6, Section 7 outlines the related work and Section 8 summarizes this article.

II. A MOTIVATING EXAMPLE

To further illustrate our motivations, we introduce a BPC example of Booking Travel used throughout the paper. This example is adapted from [13] and consists of two participants: *Customer* and *Travel Agency (TA)*. It allows these participants to collaboratively complete business matters related to booking travel. The BPC example represented by a BPMN collaboration model and the exchanged data are shown in Figure 1. In order to simplify the model and data flow, we present the input and output of each node in the form of data object collections and data associations.

Initially, the customer requests a travel offer from the travel agency according to the travel *period*, *departure* and *destination* information. When the *offer* is received, the customer begins to evaluate. If the *evaluation* is to reject, customer informs the travel agency of the *reason*, otherwise it will send a *confirmation* to the travel agency. Finally, the customer receives the *itinerary*. Likewise, when a travel request is received, the travel agency will make the offer, including the *hotel*, *flight* and *quote*, and then send the offer to the customer. After that, collaboration will depend on the message from the customer. If a *rejection* from the customer is received, the collaboration will end, otherwise it will send the itinerary to the customer. All the above data items need to be exchanged

between participants through messages, and they are regarded as exchanged data and presented in a global data contract by TraDE.

In this scenario, accurate message communication and data exchange are critical for successful collaboration. In fact, BPMN standard provides the concept of correlation (see Definition 3 in Section 3) to allow messages to be correlated to appropriate participant instances. Thus, on the one side, the sender first specifies the content data of the message, and appends the specific correlation data to the message, then passes the message to the receiver in the form of message flows. On the other side, the receiver first matches the correlation data of the message to receive the message, and then extracts the content data to the predefined data objects [4]. In this context, the exchanged data has to be accurately defined, used and passed among *sender*, *receiver* and *messages*. That is to say, these three aspects have to satisfy the necessary data dependencies to send or receive messages.

However, the definition, use, and delivery of exchanged data is still an error-prone task for participants. Indeed, each participant of the overall BPC develops its part independently as a service, including its own communication part and data flow. The exchanged data involving sender, receiver and messages is manually added and maintained by participants. As a result, it is easy to cause data dependency violations during collaboration and interaction, resulting in data flow errors across processes.

For example, in Figure 1, if the *Customer* process does not define *D2* into the input data of node *1* (i.e., *D2*(*Departure*) is not available in node *1*), then node *1* will not be able to meet the data required to send the *Travel Info* message because the message has been defined to contain *D1*, *D2*, *D3*, etc. Conversely, if the *Travel agency* process does not define *D3* into the output data of node *10*, then it will not able to receive the *Travel Info* message because node *10* cannot extract *D3* from *Travel Info* message to its output data and *D3* will be regarded as undefined in the *travel agency* process.

III. PRELIMINARIES

In this section, we present several definitions related to our method, some of which are extended from BPMN [14], highlighting data and message communication. Meanwhile, we introduce some auxiliary functions to facilitate the subsequent data flow processing.

Definition 1 (Business Process, BP): Generally, a business process describes a sequence of activities in an organization with the goal of carrying out work. Here, we describe the process simply as a 3-tuple $BP = (G, Input, Output)$:

(1) G: a process is depicted as a graph (N, E), where $N = N^A \cup N^E \cup N^G$ is a set of nodes that represents Activities (N^A) Events (N^E) and Gateways (N^G); and $E \subseteq (N \times N)$ is a set of directed edges (sequence flows) that define sequential relation between nodes.

(2) $Input, Output: N \rightarrow 2^D$ are functions that assign input and output data items to nodes in N, where D is the set of data items defined or used in BP, and 2^D refers to the power set of D.

Here, we use Definition 1 to represent a business process, where (1) is adapted from the BPMN standard, while (2) highlights the input and output data of nodes. In addition, we introduce an auxiliary function *Send*(*BP*) (resp. *Receive*(*BP*)) to obtain the set of nodes that sends (resp. receives) messages in BP. Moreover, we assume that each sending (resp. receiving) node can only send (resp. receive) one message.

Definition 2 (Message, M): M is a set of all messages presenting the content of all potential communication between processes within a BPC.

Definition 2 is derived from the BPMN standard. A message represents the content of a communication between two processes. The auxiliary function *ContentData:* $M \rightarrow 2^D$ assigns the content data to the messages.

Definition 3 (Correlation): Correlation is used to associate a particular message to an ongoing conversation between two particular process instances.

Specifically, the concept of *Correlation* describes a set of data items on a message that need to be satisfied in order for that message to be associated to a particular sending or receiving node [14]. In addition, the auxiliary function *CorrelationData:* $M \rightarrow 2^D$ assigns the correlation data to the messages. Note that, BPMN standard allows the content data of messages to be used for correlation purposes, and in this article, we also support other data to be used for the correlation data of messages. In short, according to Definition 2 and 3, a message m contains not only content data, but also correlation data.

Definition 4 (Business Process Collaboration, BPC): A BPC is a composition of multiple processes, in which processes can collaborate and interact with each other in the way of message communication.

(1) Structure: $BPC = BP_0 \,||\, \ldots \,||\, BP_{|\mathrm{BPC}|-1}$, where '||' represents a parallel operator, |BPC| represents the number of BPs in the BPC.

(2) Message communication relation $\Delta \subseteq \mathrm{Ns} \times \mathrm{M} \times \mathrm{Nr}$: let $\delta = (n_\mathrm{s}, \mathrm{m}, n_\mathrm{r}) \in \Delta$, where $\sqcap\_1(\delta) = n_\mathrm{s} \in Send(BP_\mathrm{i})$, $\sqcap\_2(\delta) = m \in \mathrm{M}$, and $\sqcap\_3(\delta) = n_r \in Receive(BP_\mathrm{j})$, $i \neq j$, $BP_i, BP_j \in BPC$.

(3) Exchanged data (ED): $\mathrm{ED} \subseteq \cup_{i<|BPC|} D_i$, where D_i is the set of data items defined or used in BP_i.

In this definition, (1) shows that BPC is composed of several BPs, which are parallel in the form of composition. This means that BPC is a modular and loose structure, which is convenient for composition and configuration on demand. (2) specifies message communication relation between processes, which is characterized by the sending node, receiving node and messages. Note that, we use the classical projection function $\sqcap$_i to represent the *i-th* entry of such a triple in Δ. (3) represents data that need to be exchanged between processes. It indicates a common, consolidated and agreed set of data items representing a data contract between participating processes [11], [12].

In addition, in order to facilitate the processing of messages corresponding to nodes, we give the auxiliary function $Rmsg(n_r) = \{\sqcap\_2(\delta)|\delta \in \Delta$, and $\sqcap\_3(\delta) = n_r\}$ to represent the message m received by node n_r. Similarly, function $Smsg(n_s) = \{\sqcap\_2(\delta)|\delta \in \Delta$, and $\sqcap\_1(\delta) = n_s\}$ represents the message sent by node n_s.

Note that, since we focus on data flow errors across processes, we mainly care about exchanged data, while ignoring the data irrelevant to message communication within the process. Meanwhile, we give two assumptions to avoid ambiguity. One is to assume that there is no control flow error in BPC. The other is to assume that there is no data flow error inside a single process in BPC.

IV. DATA DEPENDENCIES IN BPC

In this section, we first analyze the data dependencies between processes and messages. Then, we introduce a data model and a set of properties to formalize data dependencies.

A. DATA DEPENDENCIES

Data dependencies are derived from the data specification. The operational data specification in this article is based on the literature [8] (see (1)), while considering the message communication and data exchange requirements in BPC, as shown in (2).

(1) In a *BP*, a data item has to be defined before it can be used. When node n reads a data item di, di is part of its input data, meaning that it is used by node n. When node n performs a write operation on di, di is the part of its output data, meaning that it is defined or redefined by node n.

(2) In a *BPC*, each *BP* also needs to consider the impact of external data flows, that is, the data flows related to messages.

(2.1) **Between receiving nodes and messages**: when receiving a message, the node needs to go through two steps. Firstly (*matching step*), **it has to accurately define input data to match the *correlation data* of message**. Secondly (*transformation step*), **it has to accurately define output data where the *content data* of message should be extracted to**. According to (1), data needs to be defined or redefined. The receiving node has to extract the content data of message to its output data (i.e., write operation). Otherwise, the data will be considered undefined and cannot be used by the receiver.

(2.2) **Between sending nodes and messages**: when sending a message, **the node has to accurately define input data and message related data, so that the available input data can match (fill in) the content and related data of the message.** Otherwise, the message cannot be sent due to the lack of necessary data.

(2.3) **Between messages and processes**: **the correlation data of all messages received by the same process are not empty**. Indeed, many instances of a particular process may typically run in parallel [14], if the correlation data of the messages is empty, any process instance can match it, resulting in message confusion.

Note that in some simple scenarios, (2.3) is optional, but considering the complexity of data flow design and maintenance, and the relative independence of multiple participants in BPC, we consider it mandatory from a unified and more strict perspective.

B. DATA MODEL AND PROPERTIES

The data model is applied to each process node. It not only intuitively shows the input and output data of nodes within the process, but also explicitly captures the data flows related to messages.

Definition 5 (Data Model): A data model of node n is a 4-tuple $DM = (Input, Output, ContentSet, CorrelationSet)$:

(1) *Input*: the input data of node n
(2) *Output*: the output data of node n
(3) *ContentSet*: the content data of the message associated with node n
(4) *CorrelationSet*: the correlation data of the message associated with node n

Specifically, $Input(n)$ and $Output(n)$ are defined in Definition 1, while $ContentSet(n)$ and $CorrelationSet(n)$ are derived from Definition 2, 3, that is to say:

If $n \in Send(BP)$, then $ContentSet(n) = ContentData\ (Smsg(n))$, and $CorrelationSet(n) = CorrelationData\ (Smsg(n))$

If $n \in Receive(BP)$, then $ContentSet(n) = ContentData\ (Rmsg(n))$, and $CorrelationSet(n) = CorrelationData\ (Rmsg(n))$

To better understanding of data model, Figure 2 shows the fragment of the data model applied to Booking Travel example. Meanwhile, the complete data model information of this example is shown in Table 1. Note that all four constituent elements of the data model are optional. For example, the data model of node *2* is not empty, while that of node *13* is empty. In particular, a data based XOR gateway (node *4*) usually has only input data to decide which branch to take.

In Figure 2, the task of node *1* ($Send\ Travel\ Request \in Send(Customer)$) is to send a message to *travel agency* process. It uses the data items of $Input(1)$ to fill in the content data ($ContentSet(1)$) and correlation data ($CorrelationSet(1)$). Then, the Node *2* ($Receive\ Offer \in Receive(Customer)$) receives a message from *travel agency* process. It uses the data items of $Input(2)$ to match the correlation data ($CorrelationSet(2)$), which determines whether it can receive the message. When the message is successfully matched, the content data ($ContentSet(2)$) is extracted to $Output(2)$. Node *3* does not involve sending or receiving message, so it has only $Input(3)$ and $Output(3)$, while its $ContentSet(3)$ and $CorrelationSet(3)$ are empty.

TABLE 1. Complete data model information for Booking Travel example.

BPC	Node	Input data	Output data	ContentSet	CorrelationSet	Annotation
Customer (C)	Start	∅	D0, D1, D2, D3, D4	∅	∅	Initial data
	1	D0, D1, D2, D3, D4	∅	D0, D1, D2, D3	D4	Travel Info Message
	2	D0, D1, D2, D3	D5, D6, D7	D5, D6, D7	D0, D1, D2, D3	Offer Message
	3	D5, D6, D7	D8	∅	∅	Non-communication node
	4	D8	∅	∅	∅	Data based XOR gataway
	5	D4, D8	D9	∅	∅	Non-communication node
	6	D4, D8	D10	∅	∅	Non-communication node
	7	D0, D5, D6, D7, D8, D9	∅	D8, D9	D0, D5, D6, D7	Rejection Message
	8	D0, D5, D6, D7, D8, D10	∅	D8, D10	D0, D5, D6, D7	Confirmation Message
	9	D0, D10	D11	D11	D0, D10	Itinerary Message
Travel Agency (TA)	Start	∅	D4	∅	∅	Initial data
	10	D4	D0, D1, D2, D3	D0, D1, D2, D3	D4	Travel Info Message
	11	D0, D1, D2, D3	D5, D6, D7	∅	∅	Non-communication node
	12	D0, D1, D2, D3, D5, D6, D7	∅	D5, D6, D7	D0, D1, D2, D3	Offer Message
	13	∅	∅	∅	∅	No data involved
	14	D0, D5, D6, D7	D8, D10, D11	D8, D10	D0, D5, D6, D7	Confirmation Message
	15	D0, D5, D6, D7	D8, D9	D8, D9	D0, D5, D6, D7	Rejection Message
	16	D0, D10, D11	∅	D11	D0, D10	Itinerary Message

According to data dependencies analyzed in 4.1, we can give the following properties that need to be satisfied in collaboration with the help of data model.

Property 1 (Corresponding to Section 4.1 (2.2)): If $n \in Send(BP)$, then $Input(n) \supseteq (ContentSet(n) \cup CorrelationSet(n))$

Property 1 specifies the condition that has to be met between the sending nodes and the messages, i.e., the data dependencies between input data and message-related data (content data and correlation data).

Property 2 (Corresponding to Section 4.1 (2.1)): If $n \in Receive(BP)$, then $(Input(n) \supseteq CorrelationSet(n)) \wedge (Output(n) \supseteq ContentSet(n))$

Property 2 specifies the condition that has to be met between the receiving nodes and the messages, i.e., the data dependencies between input data and correlation data, as well as the data dependencies between output data and content data.

Property 3 (Corresponding to Section 4.1 (2.3)): If $BP \in BPC$, then $\forall m \in \cup_{n \in Receive(BP)} Rmsg(n)$: $CorrelationData(m) \neq \emptyset$

Property 3 specifies the data dependencies that all messages received by a process must comply with.

V. DETECTING DATA FLOW ERRORS IN BPC

In this section, we formally define a set of data flow error patterns across processes. Moreover, we propose an algorithm to detect them automatically.

A. DEFINITIONS OF DATA FLOW ERROR PATTERNS

Definition 6 (Unable to Send Message in BPC, USMB): In process *BP* of *BPC*, node *n* involves a USMB if its input data is not accurately defined, it cannot fill in the content data and correlation data of the message it sends. More formally, in process *BP* of *BPC*, node *n* involves a USMB if the following holds:

$$n \in Send(BP) \wedge (Input(n) \not\supseteq (ContentSet(n) \cup CorrelationSet(n)))$$

For example, in node *12* (see Figure 1), the travel agency sends the offer to the customer through the *Offer* message. The correlation data specified in the message contains *D0-D3* (see Table 1). However, if the *D0* is not defined in the input data of node *12*, it will not be able to fill in the correlation data. As a result, the message could not be sent. Similarly, if the content data of the *Offer* message contains *D4* in addition to *D5, D6* and *D7*, node *12* is also unable to send the message because *D4* is not defined in *Input* (*12*). In short, this error conflicts with Property 1.

Definition 7 (Unable to Receive Message in BPC, URMB): In process *BP* of *BPC*, node *n* involves a URMB if its input data is not accurately defined, it cannot match the correlation data of the message it receives, or its output data is not accurately defined, then it cannot extract the content data of the message it receives. More formally, a process *BP* of *BPC*, node *n* involves a URMB if the following holds:

$$n \in Receive(BP) \wedge ((Input(n) \not\supseteq CorrelationSet(n)) \vee (Output(n) \not\supseteq ContentSet(n)))$$

For example, in node *2* of *Customer* process, the customer receives an offer from the travel agency. The correlation data specified in the *Offer* message are *D0-D3*. However, if *D0-D3* are not defined in *Input*(*2*), or one of them is missing, node *2* cannot match the message. Similarly, if there is *D2* in the content data of the *Offer* message, node *2* cannot receive the message because *D2* is not defined in *Output*(*2*), that is, the node *2* cannot properly extract the content data of the message to its output data. In short, this error conflicts with Property 2.

Definition 8 (Unable to Match Correct Message in BPC, UMCMB): A process *BP* involves a UMCMB if at least one of the messages received by *BP* has no correlation data. More formally, process *BP* in *BPC* involves a UMCMB if the following holds:

$$\exists m \in \cup_{n \in Receive(BP)} Rmsg(n) : CorrelationData(m) = \emptyset$$

For example, according to BPMN standard, many instances of a particular process will typically run in parallel. *Correlation* is used to associate a particular message to a particular process instance. Specifically, there may be multiple instances of both the *Customer* process and the *Travel Agency* process running at the same time. If the correlation data of a message is empty, any receiver process instance can match it, resulting in message confusion. In short, this error conflicts with Property 3.

B. ALGORITHM FOR DETECTING DATA FLOW ERRORS

According to the above definitions and properties, we propose an algorithm to automatically detect data flow errors. Here, we take the detecting USMB as an example for algorithm analysis, while the complete algorithm, covering detecting URMB and UMCMB, is similar to the former, and is shown in the form of appendix in [15].

Part 1 of algorithm is presented to detect all nodes with USMB error. Its core is to use Property 1 and Definition 6 to detect the sending nodes. It determines whether the node has accurately defined its input data to be able to fill in the content data and correlation data of the message. Specifically, for each process, Part 1 of algorithm first finds all sending nodes. Then, it counts the content data and correlation data of message related to the node. Finally, it determines whether the node satisfies the Definition 6.

Part 1 of Algorithm Detecting Data Flow Errors-USMB

Input info: *BPC*
Output info: *OutNodeSet* /*The set of nodes with USMB error*/
1: *OutNodeSet* $\leftarrow \emptyset$
2: **for** each $BP \in BPC$ **do**
3: **for** each node $n \in Send(BP)$ **do** /*According to Definition 6*/
4: $ContentSet(n) = \cup_{m \in Smsg(n)} ContentData(m)$,
5: $CorrelationSet(n) = \cup_{m \in Smsg(n)} CorrelationData(m)$
6: **if** $Input(n) \not\supseteq (ContentSet(n) \cup CorrelationSet(n))$ **then**
7: $OutNodeSet \leftarrow OutNodeSet \cup \{n\}$
8: **end if**
9: **end for**
10: **end for**
11: **return** *OutNodeSet*

Next, we briefly analyze the time complexity of Part 1 of algorithm. Suppose a BPC has |BPC| processes, each of which contains |N| nodes and |D| data items. Part 1 needs to obtain sending nodes and corresponding data items in each process. Therefore, in the worst case, its time complexity is O(|BPC| × |N| × |D|).

VI. EVALUATION

In this section, we report two evaluations. The first evaluation, as a benchmark study, explores whether our method can correctly detect data flow errors and whether it causes misdiagnoses. The second evaluation, as a comparative study, compares our method with the latest methods in accuracy and performance when given the same data set. All evaluations have been carried out on a PC running Windows 10 (64 bits), and it has a 3.00 GHz processor and 8 G RAM.

A. BENCHMARK EVALUATION

This evaluation consists of three stages from designing a set of collaboration models with data items, adding data flow errors into some of collaboration models, to conducting a user experiment for modeling data in collaboration models. The stage 1 serves as the basis for subsequent stages, while the stage 2 and stage 3 are used to obtain the test models and illustrate the effectiveness of our method.

In stage 1, we invite experts and our partners to design collaboration models for five scenarios, namely Booking Travel (S1), Crowdsourcing (S2), Paper Review (S3), Pastry Cook (S4) and Online Education (S5). These scenarios are adapted from existing literature [13], BPMN example library [14] as well as real-world business cases from our partners. These models use a lot of data information. To reduce unnecessary interferences, we have confirmed in advance that they have no data flow errors and control flow errors (all of them are available in [15]).

In fact, in order to obtain collaboration models, we first planned to select models from publicly available model libraries as benchmark, such as BPMN example library. Unfortunately, only a small fraction of the models contains data items. Moreover, many models still have problems such as lack of necessary data annotations, unclear data identifications, and even control flow errors. To deal with this problem, we invited the experts and our partners to provide the above collaboration models. These models come from different areas and case libraries, which can reflect the diversity and realistic complexity of collaboration scenarios.

In stage 2, we ask experts and our partners to add data flow errors to some collaboration models (6 errors in total, 2 in *S1.2*, 2 in *S3.2*, and the rest in *S4.2*), while other models (*S2.1* and *S5.1*) do not add errors, as shown in Table 2. In this way, we obtain a set of models (8 in total) that covers all the error patterns we defined.

In stage 3, we conduct a user experiment to get a set of collaboration models that might have data flow errors to further evaluate our method. Unlike the stage 2, we do not know the possible errors in the models in advance, thus this provides us the opportunity to analyze data flow errors using statistics. The experiment involves 30 graduate students with BPMN modeling knowledge, including data aspect.

TABLE 2. Statistics of collaboration models in stage 2. *Sx.1* means the original model without data flow errors for scenario *Sx*, any other *Sx.n* represents the *n-th* variant of a model with errors.

Model		USMB	URMB	UMCMB
Booking Travel	S1.1	0	0	0
	S1.2	0	1	1
Crowdsourcing	S2.1	0	0	0
Paper Review	S3.1	0	0	0
	S3.2	1	1	0
Pastry Cook	S4.1	0	0	0
	S4.2	1	0	1
Online Education	S5.1	0	0	0
	Total	2	2	2

Specifically, we describe the 5 models obtained from stage 1 and data specifications in the form of text, so that participants can model the data flow for these models. We remove the data items from the models and assign the models to different groups (6 participants per group). All participants are required to independently complete the data modeling of a given model and, if necessary, allow them to modify the model.

Finally, we obtain 33 collaboration models with data items in stage 3, in which some participants provide multiple variants of the same model. These models differ in the number of exchanged data and elements (e.g., communication nodes and gateways) because the models have been modified by the participants as appropriate. Moreover, we manually check whether these models have data flow errors. In the end, we find a total of 29 errors, including 9 USMB, 14 URMB and 6 UMCMB. This can be regarded as the gold standard for evaluating our method. Due to space limitation, complete model statistics are displayed in [15], while refined statistics are shown in Tables 3.

TABLE 3. Statistics of the data flow errors in the 33 collaboration models.

Model set	Type of error	Number of models	Percentage (%)
Models with errors (18)	USMB	9	27.27
	URMB	12	36.36
	UMCMB	6	18.18
Models without errors (15)	-	-	45.45

Results: We firstly evaluate our method using the models obtained in stage 2 and count the corresponding results. The evaluation of stage 2 (6 errors in 8 models) show that all data flow errors added in advance have been detected accurately, and no false positives have been triggered, including those models without errors.

Secondly, as a further extension and generalization of stage 2, we conduct the evaluation based on stage 3 (29 errors in 33 models). By comparing with the errors checked by hand, we find that all data flow errors in stage 3 have been detected, and there are no misdiagnoses. These two evaluations indicate that our method is effective for all data flow error patterns we defined.

Table 3 summarizes the frequency of each data flow error pattern in the 33 collaboration models. More than half of the models ($18/33 = 54.55\%$) have data flow errors, and some of them have different types of errors. URMB is the most common data flow error pattern, with 12 ($12/33 = 36.36\%$) models showing this type of error. In particular, two of these models contain two such errors. The proportion of USMB errors is about one third ($9/33 = 27.27\%$) while that of UMCMB is $6/33 = 18.18\%$. In short, it indicates that data flow errors across processes remain a serious problem, which reminds us that we need to pay more attention to prevent and avoid these errors, especially URMB.

B. COMPARATIVE EVALUATION

The second evaluation explores the differences in accuracy and performance between our method (i.e., based on data flow error patterns in BPC (DEB)) and the latest methods (i.e., based on model checking (MC) [5] and data flow simulation (DFS) [4]). The reason for choosing them is that they represent state of the art and are based on BPMN modeling like DEB. We use the two model sets obtained in the first evaluation as test data. Since there is no significant difference between the results of the two data sets in statistics, we mainly show and analyze the results of the first data set (8 models). Due to some models may contain multiple errors, we record the cumulative time required to detect all errors. Meanwhile, each cycle in the model is defined to run only once.

Table 4 summarizes the statistics for each method in the comparative evaluation. We list the core elements of the model as indicators of their complexity, in which *S_node*, *R_node*, *Gateway* and *Data item* represent the number of sending nodes, receiving nodes, gateways and exchanged data in the model, respectively.

We find that these three methods can meet the expected goal in terms of accuracy, all of which are 100%, but they show differences in performance (i.e., DEB > MC > DFS). There are following reasons for this.

(1) MC provides a framework based on modeling and model transformations to verify behavioral properties related to BPMN collaboration model. It first needs to perform two consecutive model transformations on the initial model (i.e., from BPC to RECATNet, and from RECATNet to rewriting logic). Then it uses the Maude LTL model checker that supports rewriting logic to verify custom behavior properties. As a result, these consecutive transformations and the state space traversal of the whole model consume a lot of cost and time, making performance of MC sensitive to the size of model (the number of nodes, gateways and data items), especially when the model has parallel gateways. In S3.2, MC spends the most time, reaching 30542ms.

(2) DFS introduces a tool called Mida to simulate the execution of BPMN collaboration model, which supports the design and detection of data flow in a graphical way. Although users can monitor the execution in real time during

TABLE 4. Statistics in comparative evaluation.

Model	S_node	R_node	Gateway	Data item	Error	MC		DFS		DEB	
						Time (ms)	Accuracy (%)	Time (ms)	Accuracy (%)	Time (ms)	Accuracy (%)
S1.1	5	5	2	12	0	8143	100%	32000	100%	470	100%
S1.2	5	5	2	14	2	17944	100%	56100	100%	920	100%
S2.1	3	3	2	8	0	9437	100%	28560	100%	240	100%
S3.1	4	4	5	11	0	16641	100%	38140	100%	318	100%
S3.2	4	4	5	9	2	**30542**	100%	**85862**	100%	862	100%
S4.1	6	6	1	18	0	9157	100%	27853	100%	604	100%
S4.2	6	6	2	21	2	18785	100%	55631	100%	**1135**	100%
S5.1	5	5	0	13	0	9764	100%	22015	100%	684	100%

each simulation, the identification of data flow errors is still in charge of users, rather than through automation. In particular, when the model itself includes branch gateways, DFS requires users to manually adjust data to meet different branch conditions. For the sake of fairness, we evaluate DFS in the worst case, that is, each branch in the XOR gateway will be executed. As a result, it needs more cost and time to detect errors. With respect to performance, DFS is weaker than other methods.

(3) DEB has the best performance in this evaluation. The reason is twofold. First, DEB only focuses on the subset of the model (i.e. sending/receiving nodes, messages and exchanged data), so it does not need to perform semantic transformation and state space traversal on the whole model. Second, DEB provides a set of clear data flow error patterns and resolution algorithm, which support automatic error detection without manual intervention. In this way, it avoids the possible state space explosion and greatly reduces the detection time. From this point of view, it may be an effective supplement to the soundness analysis of data flow in BPC, and may improve user satisfaction when using interactive tools (e.g., IDE (Integrated Development Environment)).

VII. RELATED WORK

In this section, we review the related work to detecting data flow errors in a single process (DfSP) and BPC.

A. DETECTING DATA FLOW ERRORS IN A SINGLE PROCESS

In the early work, Sadiq *et al.* [16] for the first time identify a variety of data flow errors at the conceptual level, including missing data, redundant data, lost data, etc. However, it does not provide a concrete solution for data flow errors. Subsequently, a large number of methods have been proposed to analyze and detect data flow errors based on different modeling perspectives, such as BPMN [17], [18], Petri net [9], UML activity diagrams(UML AD) [8], BPEL/WS-BPEL [10], etc.

The existing DfSP focuses the data dependencies among nodes within a single process (e.g., input and output of nodes), without considering the data related to messages.

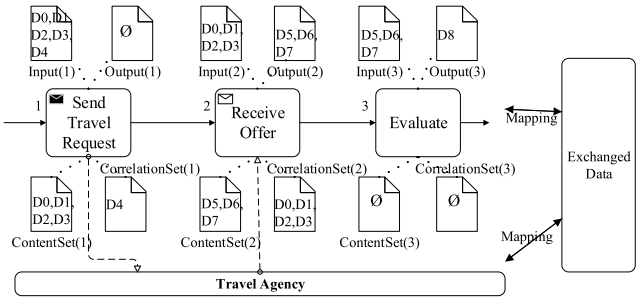

FIGURE 2. An example fragment of the Customer process with our defined data models.

This makes it unable to directly support the message communication and data exchange that BPC needs to cover.

B. DETECTING DATA FLOW ERRORS IN BPC

Recently, the work of considering data flow errors across process has attracted more and more attention. Kheldoun *et al.* [5] propose a formal method to verify BPMN collaboration models. Firstly, it transforms BPMN collaboration model, containing messages, data objects, etc., into a High-level Petri net (RECATNet), using a specific ATLAS transformation language. Then, it continues to transform the semantics of RECATNet to the conditional rewriting logic. Finally, it uses the Maude LTL model checker, which supports rewriting logic, to detect custom behavior properties. Overall, it has strict formal semantics and can support the verification of custom properties. Different from this, Corradini *et al.* [4] introduce a simulation tool to visualize data flow during BPMN collaboration execution. It analyzes the interdependencies among processes, messages and data, and provides direct formal semantics based on LTS for core components (e.g., messages and gateways). In this way, users can debug data flow errors, such as wrong data format, message-related data errors, data-based gateway errors, etc. Unfortunately, these methods rarely consider providing formal definitions for data flow errors across processes, but either implicitly represents them in user-defined data formulas or require users to be responsible for identifying them.

In addition, Yu *et al.* [6] map the collaboration model to E-Commerce Business Process Net based on Petri net to

analyze data flows among multiple participates, and propose a incidence matrix method to check data properties. Yet, it is limited to the field of E-commerce. To avoid data flow issues, Meyer *et al.* [7] extend data objects by means of text annotation to meet the data dependencies in BPMN collaboration. Unlike our work, it introduces SQL query to operate input and output data, tending to automate data processing.

Another related work is about data modeling in BPC. The authors of [19] emphasize automating data exchange between processes use messages. Furthermore, Hahn *et al.* [11] introduce TraDE to improve the flexibility of data modeling, supporting data exchange and runtime data change across processes. All cross process data items are exposed by participants in a web accessible way (e.g., through a REST API) [12]. Meanwhile, each data item is represented as a resource, so it can be easily accessed, referenced and shared with others through the uniform resource locator. In short, this method focuses on modeling data flows across processes, alleviating data heterogeneity (such as data structure and semantic differences). On this basis, we further explore the data dependencies between processes and messages to detect data flow errors.

VIII. CONCLUSION

In this article, we focus on detecting data flow errors across processes in BPC. As a core contribution, we define a set of data flow error patterns by analyzing and formalizing data dependencies in order to clearly clarify and identify errors. Meanwhile, we believe that these patterns are a foundation and beginning, and surely as more complex collaboration scenarios are explored, more new patterns and methods are bound to appear and provide power for the study of BPC. Furthermore, we develop an automatic detection algorithm, which centers on the subset of sending and receiving nodes, messages and exchanged data, without having to consider the whole model. Finally, we conduct two evaluations, including a benchmark evaluation used to explore the effectiveness of the method, and a comparative evaluation used to illustrate the difference in accuracy and performance between the method and the latest methods. The evaluation results indicate that our method can accurately detect these errors without producing false positives. In addition, all the compared methods can achieve the expected goal in accuracy, while our method is superior in performance to others.

In the future, we will continue to enrich data flow error patterns by analyzing more collaboration scenarios. In addition to the common read and write operations in data specification, we are interested in further exploring the impact of data destruction operation on data dependencies.

REFERENCES

[1] B. Fleaca, "Organization and business environment collaborative model to increase the innovation capacity," in *Proc. Int. Conf. Knowl.-Based Org.*, 2018, pp. 296–301.
[2] L. Xu and P. T. de Vrieze, "Supporting collaborative business processes: A BPaaS approach," *Int. J. Simul. Process Model.*, vol. 13, no. 1, pp. 57–72, 2018.
[3] J. Köpke and J. Eder, "Equivalence transformations for the design of interorganizational data-flow," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2015, pp. 367–381.
[4] F. Corradini, C. Muzi, B. Re, L. Rossi, and F. Tiezzi, "Animating multiple instances in BPMN collaborations: From formal semantics to tool support," in *Proc. Int. Conf. Bus. Process Manage.*, 2018, pp. 83–101.
[5] 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.
[6] W. Yu, C. Yan, Z. Ding, C. Jiang, and M. Zhou, "Analyzing E-commerce business process nets via incidence matrix and reduction," *IEEE Trans. Syst., Man, Cybern. Syst.*, vol. 48, no. 1, pp. 130–141, Jan. 2018.
[7] A. Meyer, L. Pufahl, D. Fahland, and M. Weske, "Modeling and enacting complex data dependencies in business processes," in *Proc. 11th Int. Conf. Bus. Process Manage.*, 2013, pp. 171–186.
[8] S. X. Sun, J. L. Zhao, J. F. Nunamaker, and O. R. L. Sheng, "Formulating the data-flow perspective for business process management," *Inf. Syst. Res.*, vol. 17, no. 4, pp. 374–391, Dec. 2006.
[9] N. Trčka, W. M. Van der Aalst, and N. Sidorova, "Data-flow anti-patterns: Discovering data-flow errors in workflows," in *Proc. Int. Conf. Adv. Inf. Syst. Eng.*, 2009, pp. 425–439.
[10] W. Song, C. Zhang, and H.-A. Jacobsen, "An empirical study on data flow bugs in business processes," *IEEE Trans. Cloud Comput.*, early access, Jun. 5, 2018, doi: 10.1109/TCC.2018.2844247.
[11] M. Hahn, U. Breitenbucher, F. Leymann, M. Wurster, and V. Yussupov, "Modeling data transformations in data-aware service choreographies," in *Proc. IEEE 22nd Int. Enterprise Distrib. Object Comput. Conf. (EDOC)*, Oct. 2018, pp. 28–34.
[12] M. Hahn, U. Breitenbücher, O. Kopp, and F. Leymann, "Modeling and execution of data-aware choreographies: An overview," *Comput. Sci. Res. Develop.*, vol. 33, nos. 3–4, pp. 329–340, Aug. 2018.
[13] F. Corradini, F. Fornari, A. Polini, B. Re, and F. Tiezzi, "A formal approach to modeling and verification of business process collaborations," *Sci. Comput. Program.*, vol. 166, pp. 35–70, Nov. 2018.
[14] OMG. (2011). *Business Process Model and Notation (BPMN) Version 2.0*. [Online]. Available: http://www.omg.org
[15] *exampleProcess*. Accessed: May 14, 2020. [Online]. Available: https://github.com/xthHub/RenWFMS/tree/master/exampleProcess
[16] S. Sadiq, M. Orlowska, W. Sadiq, and C. Foulger, "Data flow and validation in workflow modelling," in *Proc. 15th Austral. Database Conf.*, vol. 27, 2004, pp. 207–214.
[17] S. von Stackelberg, S. Putze, J. Mülle, and K. Böhm, "Detecting data-flow errors in BPMN 2.0," *Open J. Inf. Syst.*, vol. 1, no. 2, pp. 1–19, 2014.
[18] J. Mülle, C. Tex, and K. Böhm, "A practical data-flow verification scheme for business processes," *Inf. Syst.*, vol. 81, pp. 136–151, Mar. 2019.
[19] A. Meyer, L. Pufahl, K. Batoulis, D. Fahland, and M. Weske, "Automating data exchange in process choreographies," *Inf. Syst.*, vol. 53, pp. 296–329, Oct. 2015.

TIANHONG XIONG received the B.S. degree from Hohai University, Changzhou, China, in 2007, and the M.S. degree from Central South University, Changsha, in 2010. He is currently pursuing the Ph.D. degree in computer science with the School of Data and Computer Science, Sun Yat-sen University, Guangzhou, China. From 2014 to 2017, he was a Lecturer with the College of Computer Science, Hunan Institute of Technology, Hengyang, China. His current research interests include business process management, service computing, and software engineering.

MAOLIN PAN received the B.S. and M.S. degrees from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1988 and 1991, respectively, and the Ph.D. degree from Sun Yat-sen University (SYSU), Guangzhou, China, in 2017. He is currently a Lecturer with the School of Data and Computer Science, SYSU. His research interests include BPM and workflow technical and cloud native computing.

YANG YU received the bachelor's and master's degrees in computer science from the Huazhong University of Science and Technology (HUST), Wuhan, China, in 1988 and 1991, respectively, and the Ph.D. degree in computer science from Sun Yat-sen University (SYSU), Guangzhou, China, in 2007.

From 1991 to 2002, he was a Senior Engineer and a CTO with a software company. Since 2003, he became a Vice Professor with the School of Information Science and Technology, SYSU, where he has been a Full Professor with the School of Data and Computer Science, since 2011. He has published more than 60 articles. He has been a reviewer for several prestigious international conferences and journals. His research interests include workflow, service computing, cloud computing, and software engineering. He is a Senior Member of CCF and a member of ACM.

DINGJUN LOU received the B.Sc. degree from the National University of Defense Technology, in 1982, the M.Sc. degree from Tsinghua University, in 1984, and the Ph.D. degree from the University of Otago, New Zealand, in 1992. He is currently a Full Professor with the School of Data and Computer Science, Sun Yat-sen University. He has published 84 articles on graph theory and graph algorithm in international and Chinese journals. His research interests include in graph theory and graph algorithm.