# XCFG Based Data Flow Analysis of Business Processes

Shunhui Ji[1], Bixin Li[2], Pengcheng Zhang[1]
[1]College of Computer and Information, Hohai University, Nanjing, China
[2]School of Computer Science and Engineering, Southeast University, Nanjing, China

*Abstract*—**A lot of service-based business processes have been developed with the rapid growth of cloud computing. Ensuring the data flow correctness of business processes is very important, which needs capturing the data flow firstly. In this article, an XCFG(eXtended Control Flow Graph) based approach is proposed for the data flow analysis of business processes defined in BPEL specification. With XCFG modeling BPEL process, the improved equations for reaching definitions are defined to take the new characters of XCFG into consideration. Then iterative algorithm is adopted to solve the equations so that the definitions that may reach each XCFG node are computed. Case study shows the effectiveness of the proposed approach.**

*Index Terms*—**Business process; data flow analysis; XCFG; reaching definitions**

## I. Introduction

With the rapid growth of cloud computing, a lot of service-based business processes have been developed [1]. BPEL (Business Process Execution Language)[2] is a standard specification for describing service-based process businesses. It defines composite service by describing the interactions between process and partner services in some logical order. BPEL business process is error-prone for the complex characters of BPEL specification. The design and construction of data flow is especially error-prone because it's implicit in the business process. So ensuring the data flow correctness of BPEL process is very import.

Data flow testing plays an important role in ensuring the correctness of BPEL process. And data flow analysis is a prerequisite task in the data flow testing [3]. Furthermore, it also supports data flow anomalies detection [1], verification [4] and optimization [5]. It is meaningful to study the problem of data flow analysis for BPEL process.

In this article, an XCFG(eXtended Conrol Flow Graph) based approach is proposed to analyze the data flow of business processes defined in BPEL specification. Firstly, XCFG is constructed to model BPEL process precisely, so that data flow analysis can be automated. XCFG model explicitly describes the control flow of BPEL process, with annotation to XCFG elements to record data related information for data flow analysis. Then, based on the traditional reaching definitions analysis [5], improved equations are defined for the reaching definitions analysis for BPEL process, which take into consideration the newly introduced characters of XCFG compared with traditional control flow graph (CFG). And iterative algorithm is adopted to get the definitions that may reach each XCFG node.

In summary, this paper makes the following contributions:

- An XCFG based approach for data flow analysis of BPEL process is proposed, and also the detail algorithm is provided.
- The correctness analysis of our approach is given.
- Case study shows the effectiveness and the applications on anomaly detection and data flow testing.

The rest of this paper is organized as follows: section 2 introduces XCFG briefly; section 3 illustrates the approach of data flow analysis for BPEL process; section 4 gives the correctness analysis of the proposed approach; section 5 performs case study to show the effectiveness; section 6 discusses the related work and section 7 concludes the paper.

## II. XCFG Model

BPEL process draws up the workflow of service composition, in which each step is defined with an activity. In BPEL specification [2], there are two kinds of activity: basic activity (*invoke*, *receive*, *reply*, *assign*, etc.) and structural activity (*sequence*, *if*, *while*, *flow*, etc.).

XCFG is a kind of control flow graph, which can intuitively describe the workflow of BPEL process. It is defined as a quintuple $(N, E, PL, V, F)$, where $N$ is a set of XCFG nodes, $E$ is a set of XCFG edges, $PL$ is a set of partnerLinks, $V$ is a set of variables, and $F$ is the field of XCFG element.

- $N = N_B \cup N_S \cup N_E \cup N_C$. $N_B$ denotes basic activities (*receive*, *reply*, *assign*, etc.). $N_S$ has two sub types $N_{SS}$ and $N_{SE}$ to represent the start and end of sequence activities (*sequence* and *scope*). $N_E$ has two sub types $N_{ED}$ and $N_{EM}$ to represent the start and end of choice activities (*if*, *pick*, *while* and *repeatUntil*). $N_C$ has two sub types $N_{CB}$ and $N_{CM}$ to represent the start and end of concurrency activities (*flow*).
- $E = E_C \cup E_L$, where $E_C$ denotes control flow of activities (types of *Sequence*, *Choice* and *Concurrency*), and $E_L$ denotes links in *flow* activity.
- $PL$ denotes the partner services interacting with BPEL process, whose partner relationships are declared with *partnerLinks* element in BPEL.

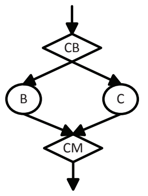
| Case | Traditional Approach | Practical Analysis | Our Approach |
|---|---|---|---|
| Out(CB)={(x,A)}<br>Gen(B)={(x,B)}<br>Kill(B)={(x,A)}<br>Gen(C)={(y,C)}<br>Kill(C)=Ø  | In(B)={(x,A)}<br>Out(B)={(x,B)}<br>In(C)={(x,A)}<br>Out(C)={(x,A),(y,C)}<br>In(CM)={(x,A),(x,B),(y,C)}<br>Out(CM)={(x,A),(x,B),(y,C)} | In(B)={(x,A),(y,C)}<br>Out(B)={(x,B),(y,C)}<br>In(C)={(x,A),(x,B)}<br>Out(C)={(x,A),(x,B),(y,C)}<br>In(CM)={(x,B),(y,C)}<br>Out(CM)={(x,B),(y,C)} | In(B)={(x,A)}<br>Out(B)={(x,B)}<br>In(C)={(x,A)}<br>Out(C)={(y,C)}<br>In(CM)={(x,B),(y,C)}<br>Out(CM)={(x,B),(y,C)} |

Figure 1. Reaching definitions of concurrency activity.

- $V$ denotes the variables defined in BPEL process.
- $F$ which is attached to XCFG elements is used to record related information. Following fields are required for our approach in particular:
  - *id*: it records the identification of node with a natural number;
  - *inEdges/outEdges*: it records the set of incoming/outgoing directed control edges of node;
  - *inLinks/outLinks*: it records the incoming/outgoing links of node;
  - *readVars/predicateVars*: it records the variables computation used (c-use)/predicate used (p-use) in the corresponding BPEL activity.
  - *writeVars*: it records the variables defined in the corresponding BPEL activity.

With XCFG definition, activities in BPEL process are described with various types of nodes and control flow among activities is described with two types of edges. The detailed definition and construction of XCFG are given in our previous works[3,6]. XCFG used in this paper is the model defined in work [3], which is the revised version of that defined in work [6]. The main improvement is that iterative activity is modeled to execute for zero time, one time and two times in XCFG model. It makes all the definition and use relationships of variables in BPEL process be covered in the XCFG based data flow analysis.

## III. Data Flow Analysis

In this section, we discuss how to analyze the data flow of BPEL process based on the XCFG model. Reaching definitions is one of the most common and useful data flow schemas [5], through which the definitions that may reach a process point along some path can be calculated.

### A. CFG Based Reaching Definitions

The traditional reaching definitions are analyzed based on CFG, where a definition $d$ is notated as $(x, n)$ to denote that variable $x$ is defined in statement corresponding to node $n$. Definition $d$ reaches $n'$ only if there is a path walking from the point immediately following $d$ to $n'$, such that $d$ is not killed along that path. The CFG based data flow equations for reaching definitions are composed of transfer equation and control-flow equation which are respectively defined as follows [5]:

$$Out(n) = Gen(n) \cup (In(n) - Kill(n)) \tag{1}$$

$$In(n) = \bigcup_{p \in pred(n)} Out(p) \tag{2}$$

where $In(n)$ and $Out(n)$ denote the definitions reaching before and after node $n$, $Gen(n)$ is the set of definitions generated by the statement in $n$, $Kill(n)$ is the set of definitions killed by the definitions in $n$, and $pred(n)$ is the set of predecessor nodes of $n$.

### B. XCFG Based Reaching Definitions

BPEL supports concurrency and synchronization dependency between concurrent activities, which cannot be depicted in CFG. The traditional equations for reaching definitions are not suitable to analyze the data flow of BPEL process. The case shown in Fig.1 is taken as example for explanation, where the first column lists the data-access information of each activity, the second and third columns show the reaching definitions computed with the traditional approach and practical analysis. In this case, two problems are found for the traditional approach:

(1) It does not take the data flow between concurrent activities into consideration. For activity $B$, it's possible that its concurrent additivity $C$ executes before $B$, so that definitions in $Gen(C)$ can reach $B$.

(2) It treats the end node of concurrency activity $CM$ in the same way with that of choice activity. With Eq.(2), only definitions in $Kill(B) \cap Kill(C)$ cannot reach $CM$. However, considering the concurrency semantic that both $B$ and $C$ will be activated, no definition in $Kill(B) \cup Kill(C)$ can reach $CM$.

With the results of the traditional reaching definitions, some data flow will be constructed while they do not exist in fact. For the case in Fig.1, if there is an activity $D$ that executes immediately after $CM$ and uses $x$, it concludes that $D$ may use the value of $x$ defined in $A$ with the traditional approach, which is not true.

To solve the reaching definitions problem of BPEL process, we improve the traditional equations to make it suitable for XCFG. Our approach is proposed on the assumption that the Bernstein Criterion [7] holds, which states that concurrent activities do not define and use the same variable. So there is no definition and use relationship between concurrent activities. The improvements mainly consider the new characters of concurrency and synchronization dependency.

Firstly, for the concurrency activity, whether the definitions killed by the enclosed child activities reach the end is analyzed. There are two situations:

- If the child activity will definitely be activated, the definitions it killed must not reach the end node of concurrency activity.
- If the child activity is activated with some condition, which means it is enclosed in a choice activity, the definitions it killed may reach the end node of concurrency activity, except when the other choice branches kill these definitions too.

Secondly, the synchronization defined by $link$ in $flow$ activity is dealt with in the same way with common edge, so that the reaching definitions of link's source activity can reach its target activity. And, link's target activity is dealt with in the same way with the end node of concurrency activity. The reason is that link's target activity can be activated until all the source activities connected through link edge and common edge are completed, which is similar to the end of concurrency activity. At the point before link's target activity, none of the definitions killed by the child activities that are enclosed in the concurrency structure and must be activated before the link's target activity will reach.

In addition, for the enclosed child activities in the concurrency structure that will be executed concurrently, data flow between them is not considered based on the Bernstein Criterion assumption.

With the above consideration, we improve the original control-flow equation Eq.(2) to Eq.(3):

$$In(n) = \bigcup_{p \in pred(n)} Out(p) - \bigcup_{m \in K} Kill(m) \qquad (3)$$

where $K$ is the set of nodes whose corresponding activities are contained in the concurrency structure and will definitely be executed before $n$ when $n$ is the end node of concurrency structure or the target node of link edge, and $K$ is $\emptyset$ for the other nodes.

## C. Data Flow Solution

There are three typical solutions to the reaching definitions[5]:

- IDEAL (Ideal Solution): it begins by finding all the executable paths leading from the program entry to the beginning of point. Then it computes the reaching definitions at the end of each possible path and applies the meet operator to find the greatest lower bound.
- MOP(Meet-Over-Paths Solution): it assumes all the paths can be taken to compute the reaching definitions for the point compared with IDEAL solution.
- MFP(Maximum Fixedpoint Solution): it simulates all possible executions by propagating definitions as far as they will go without being killed. Specifically, it iteratively computes until convergence and applies the data flow equations to every block in each iteration.

**Algorithm 1** calcReachingDefinitions()

```
Input:
    $xcfg$: the XCFG model of BPEL process;
Output:
    $In()$: the definitions reaching before each XCFG node;
 1: get the set of nodes in $xcfg$: $nodeSet$;
 2: for $node \in nodeSet$ do
 3:     identify the definitions generated by node: $Gen(node)$;
 4:     identify the definitions killed by node: $Kill(node)$;
 5:     $mustKill(node) = \emptyset$;
 6: end for
 7: identify the end node of concurrency activity and the target node of $link$:
    $reqAnlNodes$;
 8: for $node \in reqAnlNodes$ do
 9:     compute the nodes that are enclosed in the concurrency structure and execute
        before $n$ : $mustNodes$;
10:     for $node' \in nodeSet$ do
11:         if $node' \in N_{ED}$ then
12:             computes the child activities enclosed in the choice activity:
                $ifNodes$;
13:             $mustNodes = mustNode - ifNodes$;
14:         end if
15:     end for
16:     for $node' \in mustNodes$ do
17:         $mustKill(node) = mustKill(node) \cup Kill(node')$;
18:     end for
19: end for
20: initiate $In()$ and $Out()$ of each node as $\emptyset$;
21: while $Out()$ of a node changes do
22:     for $node \in nodeSet$ do
23:         $Out(node) = Gen(node) \cup (In(node) - Kill(node))$;
24:         for $node' \in pred(node)$ do
25:             $In(node) = \cup Out(node')$;
26:         end for
27:         $In(node) = In(node) - mustKill(node)$;
28:     end for
29: end while
```

The relationship of the three solutions is MFP $\leq$ MOP $\leq$ IDEAL [5], where $a \leq b$ means that $b$ is more precise than $a$. IDEAL can get the optimum solution for reaching definitions, however it is undecidable to find all executable paths. And there is no direct algorithm for the MOP solution. MFP uses the iterative algorithm to approximate the IDEAL solution conservatively. Here MFP is adopted to provide the solution to the reaching definitions for each point of BPEL process.

The process of computing the reaching definitions based on XCFG model of BPEL process is shown in Alg. 1. Firstly, the definitions generated and killed by each XCFG node should be identified. The end node of concurrency activity and the target node of $link$ which require extra analysis are stored in the set $reqAnlNodes$. For any node $n$, where $n \in reqAnlNodes$, the nodes that are enclosed in the concurrency structure and will definitely be executed before $n$ are stored in the set $mustNodes$, and all the definitions that are killed by the nodes in $mustNodes$ are stored in $mustKill(n)$. Then iterative algorithm is applied to get the solution, which is the maximum fixedpoint of the reaching definitions equations. With $In()$ and $Out()$ of each XCFG node initialed as $\emptyset$, the iteration continues until $Out()$ of each node stays unchanged, which can be judged with a boolean variable.

## IV. Correctness Analysis of Our Approach

With the improved method of reaching definitions for XCFG, there is still difference compared with the practical analysis, as shown in Fig.1. The reason is that data flow between concurrent activities is not dealt with in our approach. However, the solutions to our approach will not bring negative effects to either detecting data flow anomalies or constructing definition and use relationship. Since only $In()$ of each node is needed for later analysis, we will analyze the situations when the definitions reaching to the beginning of some node are not safe.

Applying our approach may cause the child activities enclosed in the concurrency structure have inaccurate reaching definitions. The inaccuracy mainly comes from the activity's concurrent activities and direct precedent activities within the concurrency structure. For node $n$ corresponding to such activity, there are following three situations:

1) If there exists activities that can be concurrently activated with $n$, meanwhile no direct precedent activity of $n$ exists within the concurrency structure, the value of $In(n)$ is inaccurate because it does not consider the definitions generated by the concurrent nodes.

2) If no child activity is concurrently activated with $n$, meanwhile direct precedent activities exist, the value of $In(n)$ is decided by $Out()$ of the direct precedent activities. If $Out()$ of some direct precedent activity is inaccurate, then the value of $In(n)$ is inaccurate.

3) If both the concurrent activities and direct precedent activities exist, the value of $In(n)$ may be affected by not only $Gen()$ of the concurrent activities but also $Out()$ of the direct precedent activities..

For situation 1), $n$ surely is the first activity some one concurrency branch with no $link$ targeting at $n$. The case in Fig.1 is an example that satisfies this situation, where activity $A$ is set as the target object $n$. $B$ is $A$'s concurrent activity , and there is no direct precedent child activity of $A$. With the proposed data flow approach, the definitions of $Gen(B)$ are missed in $In(A)$. On the assumption of the Bernstein Criterion, there is no definition and use relationship between $A$ and $B$. So there is no necessary to check whether the $A$ uses the definition of $Gen(B)$ in the anomaly detection. Also no definition and use relationship between $B$ and $A$ can be constructed. So the missing of $Gen(B)$ is harmless. If $A$ has other concurrent activities such as $B_1$, $B_2 ... B_k$, all the definitions generated by $B_1 \cup B_2 ... \cup B_k$ which are missed in our approach won't have any relationship with the variable used in $A$ either.

For situation 2), suppose $n'$ is the direct precedent activity of $n$, the inaccuracy of $Out(n')$ may exist in the following situations specifically:

- If there exists concurrent activity which is denoted as $n''$ but no direct precedent activity of $n'$, $Out(n')$ is inaccurate based on our approach. Since no child activity is concurrently activated with $n$, $n$ and $n''$ should either be of sequence relationship or be of choice relationship. Examples in Fig.2$(a)$ and Fig.2$(b)$ are respectively taken to illustrate how the two relationships affect the reaching definitions of the analyzed activity which is activity $A$ in the examples. In Fig.2$(a)$, there is no concurrent activity but there are direct precedent activities $B$ and $C$ of $A$. With our approach, definitions in $Gen(C)$ are missed in $In(B)$ and definitions in $Gen(B)$ are missed in $In(C)$, which means $Out(B)$ and $Out(C)$ are not accurate. However, missed $Gen(C)$ can reach along link edge $l$ and missed $Gen(C)$ can reach along edge $BtoA$ at the beginning of $A$. So the results are the same as practical analysis. In Fig.2$(b)$, with $A$'s directed precedent activity being $B$, the relationship between $A$ and $B$'s concurrent activity $C$ is choice relationship. The definitions in $Gen(C)$ are missed in $In(A)$. Since only one of $A$ and $C$ can be activated, there is no data flow relationship between $C$ and $A$. So the missing of $Gen(C)$ won't bring any negative effect.
- If there exists direct precedent activity $n''$ but no concurrent activity of $n'$, $Out(n')$ is inaccurate if $Out(n'')$ is inaccurate. The inaccuracy of $Out(n'')$ comes from either the concurrent activities of $n''$ or the concurrent activities of the precedent activities of $n''$, where $Gen()$s of these concurrent activities are missed. Since no child activity is concurrently activated with $n$, $n$ and these concurrent activities should either be of sequence relationship or be of choice relationship. If they are of sequence relationship, the missed definitions can reach $n$ through a sequence of edges finally. If they are of choice relationship, the missed definitions are not related the variable used in $n$ because the exclusive activities can not be activated at the same time. In the example shown in Fig.2$(c)$, $A$ is the analyzed activity which satisfying this situation. For $A$'s direct precedent activity $C$, $B$ and $D$ are its concurrent activity and precedent activity respectively. For $A$'s direct precedent activity $D$, $B$ is its concurrent activity. In $Out(B)$, the definitions in $Gen(C) \cup Gen(D)$ are missed. In $Out(C)$, the definitions in $Gen(B)$ are missed. However, the definitions in $Gen(B)$ and $Gen(C)$ can reach $A$ along edge $BtoA$ and link edge $l$ respectively. In addition, whether the definitions in $Gen(D)$ reach $A$ or not depends on whether they are

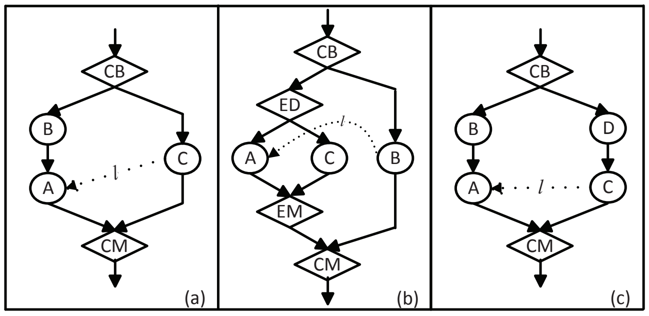


Figure 2. Correctness analysis of our approach.

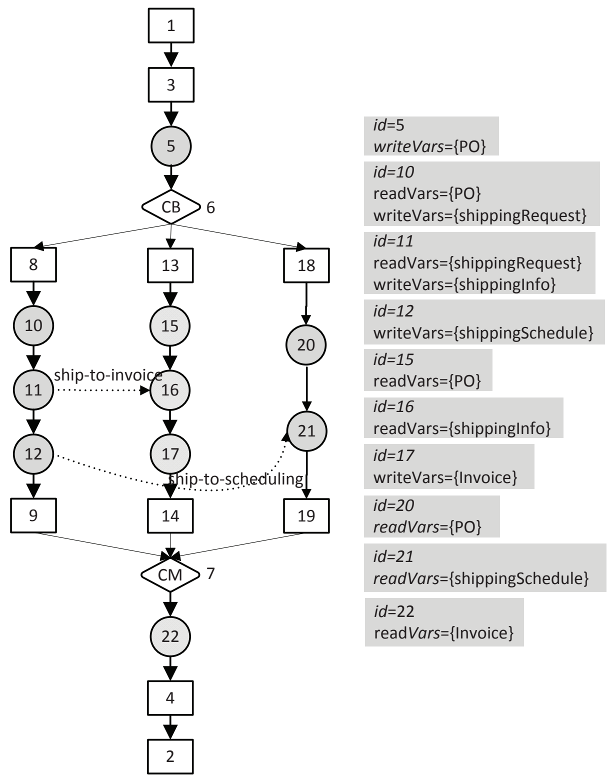


Figure 3. XCFG model of *POP*.

killed by $C$. If $C$ kills the definitions in $Gen(D)$, then the variable used in $A$ uses the definition generated in $C$ but not in $D$. So the definitions in $Gen(D)$ shouldn't reach $A$ in practical analysis either.

- If there exits both the concurrent activities and direct precedent activities of $n'$, the concurrent activities of $n'$ and $n'$'s precedent activities will make $Out(n')$ inaccurate with our approach. Also $n$ and these activities either be of sequence relationship or be of choice relationship activities, which won't affect the correct data flow construction as shown in above illustration.

For situation 3), the missing of the definitions generated by $n$'s concurrent activities does not bring negative effects as illustrated in the analysis of situation 1), and the inaccuracy caused by $Out()$ of $n$'s direct precedent activities does not bring negative effects as illustrated in the analysis of situation 2).

With the above analysis, although the reaching definitions may be inaccurate for some XCFG nodes with the proposed approach of data flow analysis, the data flow relationship can be constructed correctly based on the results of reaching definitions.

## V. Case Study

We choose the Purchase Order Process (*POP*) [2] as an example. POP is composed of one process and three partner services including *invoiceService*, *shippingService* and *schedulingService*. The process first receives the purchase order from a customer, then activates three concurrent tasks: *i)* calculating the final price for the order; *ii)* selecting a shipper; *iii)* scheduling the production and shipment for the order. There are synchronization dependencies between these concurrent tasks. Finally, the invoice is replied to the customer.

The XCFG model of POP is shown in Fig.3. The number attached to each XCFG node is its *id* filed. We also list detailed fields of some XCFG nodes in gray boxes that record accessed variables.

Applying Algorithm $calcReachingDefinitions()$ to $POP$, the reaching definitions of each XCFG node are shown in Table 1, in which node is represented with $id$ field. The results of data flow analysis are the same as manual analysis, which shows the effectiveness of our approach.

With the reaching definitions of each XCFG node, data flow anomalies can be detected and definition-use pairs can be prepared for data flow testing.

**(1) Anomaly Detection**

$D$- anomaly exists in BPEL process if a variable is defined in some activity while the value of the definition is never used in any subsequent activity. Let $nodeSet$ and $defineSet$ respectively denote the set of all nodes and the set of all definitions in XCFG model. For $\forall define \in defineSet$, it is used if and only if $\exists node \in nodeSet$ such that the following condition is satisfied:

$$define \in In(node) \wedge define.variable \in use(node)$$

where $define.variable$ denotes the defined variable, and $use(node)$ denotes the set of variables used in $node$.

*-u* anomaly exists in BPEL process if a variable is used in some activity while the variable has never been defined in any preceding activity. For each variable $var$ used in $node$, $var$ has been defined if and only if $\exists define \in In(node)$ such that the following condition is satisfied:

$$var == define.variable$$

**(2) Definition-use Pairs Computation**

For each definition $d = (x, n)$, the nodes $N$ that use definition $d$ need to be identified to construct def-use pairs $(x, n, n')$ where $n' \in N$. With the reaching definitions, $N$ can be computed with the following equation:

$$N = \{n' | (x, n) \in In(n') \wedge x \in use(n')\}$$

According to the above illustration, nor *d-* anomaly or *-u* anomaly is detected for $POP$. And def-use pairs of $POP$ are calculated, which are $(PO, 5, 10)$, $(PO, 5, 15)$, $(PO, 5, 20)$, $(shippingRequest, 10, 11)$, $(shippingInfo, 11, 16)$, $(shippingSchedule, 12, 21)$ and $(Invoice, 17, 22)$.

## VI. Related Work

Many researcher have studied the data flow analysis problem of service-based business process.

TABLE 1. REACHING DEFINITIONS OF *POP*

| node | In(node) |
|---|---|
| $6, 8, 10, 13, 15, 18, 20$ | $(PO, 5)$ |
| $11$ | $(PO, 5), (shippingRequest, 10)$ |
| $12, 16, 17$ | $(PO, 5), (shippingRequest, 10), (shippingInfo, 11)$ |
| $9, 21, 19$ | $(PO, 5), (shippingRequest, 10), (shippingInfo, 11), (shippingSchedule, 12)$ |
| $14$ | $(PO, 5), (shippingRequest, 10), (shippingInfo, 11), (Invoice, 17)$ |
| $7, 22, 4, 2$ | $(PO, 5), (shippingRequest, 10), (shippingInfo, 11), (shippingSchedule, 12), (Invoice, 17)$ |

Kopp et.al [7] presented an algorithm to derive explicit data links in BPEL process with DPE (dead path elimination) considered and showed that the approach reduces the number of derived data links compared to the approaches that ignore DPE.

Zheng et.al [8] demonstrated how to analyze the internal and external data dependencies of BPEL process with web service automata (WSA). The work addressed the shortcoming that automate based test case generation omits data dependencies.

Moser et.al [4] constructed a CSSA (Concurrent Single Static Assignment Form) representation for BPEL process to extract data dependencies, which are then used to construct a more precise formal model to improve the precision of verification. Bartolini et.al [9] studied the potential of data flow modeling of Web composite service for validation purpose, and discussed two outlined approaches to a case study, including building data flow model from requirements and extracting data flow information from BPEL.

Song et al. [1] performed empirical analysis on data flow bugs with real-world BPEL processes, where different reaching definitions are defined to detect three different kinds of data flow bugs. Yang et al. [10] constructed the control flow graph for BPEL and proposed LiveDef algorithm and AvailDef algorithm to detect data flow anomalies.

Liu et.al [11] modeled BPEL process with XBFG (eXtensible BPEL Flow Graph) and derived data dependency, which is defined as definition-use pair of operations, to evaluate the trust value of component service.

Heinze [12] argued for the idea of certified data flow analysis of business process, sketched the challenges and outlined the first steps.

The data flow analysis of BPEL process proposed in this paper is the improvement of the traditional reaching definitions, which has theoretical support and has been practically applied. The solutions to the reaching definitions not only can be used to detect data flow anomalies, but also can be used to guide the data flow testing of BPEL process [3].

## VII. CONCLUSION

The new characters of BPEL process bring challenge to analyzing the data flow. In this article, we defined the improved equations of reaching definitions based on XCFG model for the data flow analysis of BPEL process. And iterative algorithm is performed to get the solution. Although the proposed approach is imperfect, it has no negative effects on detecting data flow anomalies and constructing definition-use dependencies. Case study shows the effectiveness.

Our research represents an initial work on the data flow analysis for service-based business process. There are a lot of work to do in the future, such as considering DPE defined in BPEL, distinguishing the definition/use to parts and whole of complex typed variable, and so on.

## ACKNOWLEDGMENT

This work is supported by National Natural Science Foundation of China (No.61702159) and Natural Science Foundation of Jiangsu Province (No.BK20170893).

## REFERENCES

[1] W. Song, C Zhang, and H.A. Jacobsen. *An Empirical Study on Data Flow Bugs in Business Processes.* IEEE Transactions on Cloud Computing, PrePrints, doi:10.1109/TCC.2018.2844247.

[2] A.Alves, A. Arkin, S. Askary, et al. *Web Services Business Process Execution Language Version 2.0.* OASIS Standard, 2007.

[3] S. Ji, B. Li, and P. Zhang. *Test Case Selection for Data Flow Based Regression Testing of BPEL Composite Services.* In Proc. of IEEE International Conference on Service Computing, San Francisco, USA, 2016:547-554.

[4] S. Moser, A. Martens, K. Gorlach, et al. *Advanced Verification of Distributed WS-BPEL Business Processes Incorporating CSSA-based Data Flow Analysis.* In Proc. of IEEE International Conference on Service Computing, Salt Lake City, USA, 2007:98-105.

[5] A.V. Aho, M.S. Lam, R. Sethi, et al. *Compilers: Principles, Techniques, and Tools.* New York: Addision-Wesley, 2006.

[6] B. Li, S. Ji, D. Qiu, et al. *Verifying the Concurrent Properties in BPEL Based Web Service Composition Process.* IEEE Transactions on Network and Service Management, 2013, 10(4):410-424.

[7] O. Kopp, R. Khalf, and F. Leymann. *Deriving Explicit Data Links in WS-BPEL Processes.* In Proc. of IEEE Internatioanl Conference on Service Computing, Honolulu, USA, 2008:367-376.

[8] Y. Zheng, J. Zhou and P Krause. *Analysis of BPEL Data Dependencies.* In Proc. of 33rd EUROMICRO Conference on Software Engineering and Advance Applications, Lubeck, Germany, 2007.

[9] C. Bartolini, A. Bertolino, E. Marchetti, et al. *Data Flow-Based Validation of Web Services Compositions: Perspectives and Examples.* Lecture Notes in Computer Science, 2008, 5135:298-325.

[10] X. Yang, J. Huang, Y. Gong. *Static Data Flow Analysis and Anomalies Detection for BPEL.* In Proc. of Intenational Conference on Test and Messurement, Hong Kong, China, 2009:18-21.

[11] C. Liu, B. Li, S. Qi, et al. *Data Dependency Based Trust Evaluation for BPEL Processes.* In Proc. of 19th Asia-Pacific Software Engineering Conference, Hong Kong, China, 2012:857-866.

[12] T.S. Heinze. *Towards Certified Data Flow Analysis of Business Processes.* In Proc. of 9th Central European Workshop on Services and their Composition (ZEUS), Lugano, Switzerland, 2017.