Computing and Informatics, Vol. 39, 2020, 193–212, doi: 10.31577/cai_2020_1-2_193

CHECKING DATA-FLOW ERRORS BASED ON THE GUARD-DRIVEN REACHABILITY GRAPH OF WFD-NET

Dongming Xiang

School of Information Science and Technology
Zhejiang Sci-Tech University
310018 Hangzhou, China
e-mail: flysky_xdm@163.com

Guanjun Liu

Department of Computer Science
Key Laboratory of Embedded System and Service Computing (MOE)
Tongji University
201804 Shanghai, China
e-mail: liuguanjun@tongji.edu.cn

Abstract. In order to guarantee the correctness of workflow systems, it is necessary to check their data-flow errors, e.g., missing data, inconsistent data, lost data and redundant data. The traditional Petri-net-based methods are usually based on the reachability graph. However, these methods have two flaws, i.e., the state space explosion and pseudo states. In order to solve these problems, we use WFD-nets to model workflow systems, and propose an algorithm for checking data-flow errors based on the guard-driven reachability graph (GRG) of WFD-net. Furthermore, a case study and some experiments are given to show the effectiveness and advantage of our method.

Keywords: Petri net, workflow system, data-flow errors, reachability graph

Mathematics Subject Classification 2010: 68-Q60

1 INTRODUCTION

Nowadays, workflow systems have been widely applied to our daily life, e.g., office automation (OA), medical treatment and electronic commerce, etc. In order to guarantee the correctness of workflow systems, we not only need to verify some properties and detect errors in the control-flows, but also model and analyze their data-flows. As we know, the control-flows focus on the partial orders of business activities, while the data-flows mostly include data elements, data operations (i.e., read, write and delete) and data conditions. The existing modeling and analysis methods of workflow systems are mainly concerned with the error detection of control-flows. In fact, data-flows are also greatly important in the design of workflow system. Once its activities conduct an improper operation on data-flows in business processes, some data-flow errors [15, 19, 28] easily take place, e.g., missing data, inconsistent data, lost data, redundant data and unsoundness. These errors can lead to some abnormal results, degrade the execution performance, and increase the maintenance cost, or even result in some insecurity problems, e.g., privacy disclosure, illegal user access, and fund loss.

There have been many studies on data-flows of workflow systems. Sadiq et al. [19] first proposed seven kinds of data-flow anomalies, but did not provide any detection methods. Sharma et al. [21] used BPMN (Business Process Modeling Notation) to model business processes and detected their data-flow errors. Guo et al. [8] solved the data exchange problems in the inter-organizational workflows. Sun et al. [24] calculated the dependence relationship of business processes in a UML (Unified Modeling Language) diagram, and detected errors in each process instance according to its data association. This work was further generalized in [15], where a systematic graph traversal approach was proposed to detect data-flow errors.

Some Petri-net-based methods are also proposed to detect data-flow errors. A Dual Flow Net (DFN) [27] was used to model the control- and data-flows in an embedded system. Based on the work in [21], Awad et al. [2] mapped BPMN into Petri net, and then detected and repaired its errors. In order to model the concurrent read operation, contextual net [3, 16] was proposed, and its unfolding technique was utilized to generate the minimal test suites for multi-threaded programs [13, 12]. Based on the contextual net, PN-DO (Petri net with data operation) [31] was given to detect data-flow errors of workflow systems. All of these methods have an advantage of a great capability to specify parallelism, concurrency and synchronization [11, 17]. However, the explicitly modeling of read/write arcs can increase the scales and complexity of Petri nets. By comparison, WFD-net is a workflow net [18] (a special Petri net) extended with conceptual data operations. Its transitions are labeled by *read*, *write*, *delete* or *guard*[1] functions [10, 22]. Naturally, the scale of WFD-net is much more smaller than the Petri nets with data operation arcs, e.g., contextual net and PN-DO.

[1] A guard is a Boolean expression which is formed by some data elements and predicates.

WFD-net has been widely used to check soundness [22], completion requirements [25] and data consistencies [34]. These verification/analysis methods are usually based on the classical reachability graphs [22, 32] of WFD-nets. However, they easily suffer from the state space explosion and pseudo states. On the one hand, a state may have an exponential number of successor states in a reachability graph since every possible value of a guard is considered. On the other hand, the logical relation (e.g., exclusion property) between guards likely generate some pseudo states. In order to solve these problems, we proposed the guard-driven reachability graph (GRG) of a WFD-net in the previous work [30].

In this paper, we use GRG to check data-flow errors in a workflow system, including missing data, inconsistent data, redundant data and lost data. We first define these data-flow errors in a WFD-net, and propose an algorithm for checking them. Furthermore, a case study and some experiments are given to illustrate the effectiveness and advantage of our algorithm.

The rest of this paper is organized as follows. Section 2 presents some basic notations. Section 3 proposes an algorithm to check data-flow errors based on the GRG of WFD-nets. Section 4 gives a case study. Section 5 conducts a group of experiments. The last section sums up the whole work.

2 BASIC NOTATIONS

2.1 WF-Net

A *net* is a triple $N = (P, T, F)$, where P and T are two disjoint and finite sets that are respectively called *place set* and *transition set*, and $F \subseteq (P \times T) \cup (T \times P)$ is a *flow* relation. A net N with an initial marking m_0 is called a *Petri net* or *net system* [14, 33], and denoted as $\Sigma = (N, m_0)$. For each node $x \in P \cup T$, its *preset* and *postset* are denoted by ${}^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$, respectively.

Given a net $N = (P, T, F)$, a transition $t \in T$ is *enabled* at a marking m if $\forall p \in P : p \in {}^\bullet t \Rightarrow m(t) \geq 1$, which is denoted by $m[t\rangle$. After *firing* an enabled transition t at m, a new marking m' is generated, which is denoted as $m[t\rangle m'$, where $\forall\, p \in P$:

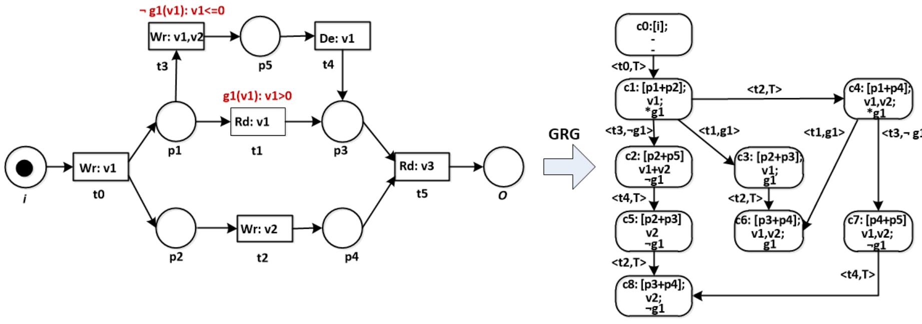
$$m'(p) = \begin{cases} m(p) - 1, & \text{if } p \in {}^\bullet t - t^\bullet, \\ m(p) + 1, & \text{if } p \in t^\bullet - {}^\bullet t, \\ m(p), & \text{otherwise.} \end{cases} \tag{1}$$

Definition 1 (Workflow net [1])**.** A net $N = (P, T, F)$ is a workflow net (WF-net) if

1. there is one source place i and one sink place o in P such that ${}^\bullet i = \emptyset \wedge o^\bullet = \emptyset$; and
2. $\forall x \in P \cup T$: $(i, x) \in F^*$ and $(x, o) \in F^*$ where F^* is the reflexive-transitive closure of F.

As a particular class of Petri net, WF-net has been widely used to model and verify workflow systems [1].

2.2 WFD-Net



a) b)

Figure 1. WFD-net and data-flow errors

Based on WF-net, *workflow net with data* (WFD-net) [25, 26] is proposed to model the control-flows and data-flows of a workflow system. That is, some data elements, data operations and guards are added into WF-net.

In a WFD-net, D is a finite set of data elements, and G is a set of *guards* over D. Var(g) denotes the set of variables in the guard $g \in G$. We assume that $\forall g \in G : d \in \text{Var}(g) \Rightarrow d \in D$. Two different guards g_1 and g_2 are *exclusive* if $g_1 = \neg g_2$ and $g_2 = \neg g_1$, which is denoted by $g_1 \otimes g_2$. In other words, g_1 is TRUE iff g_2 is FALSE, and vice versa. For example, given two guards g_1 and g_2, if $g_1(v_1) : v_1 > 0$ and $g_2(v_1) : v_1 \leq 0$, then they are exclusive and denoted by $g_1 \otimes g_2$.

Definition 2 (Workflow net with data [22])**.** A 9-tuple $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ is a WFD-net if

1. (P, T, F) is a WF-net;
2. D is a finite set of data elements;
3. G is a finite set of guards;
4. Rd: $T \to 2^D$ is a label function of reading data;
5. Wr: $T \to 2^D$ is a label function of writing data;

6. *De*: $T \to 2^D$ is a label function of deleting data; and
7. *Guard*: $T \to G$ is a label function of assigning a guard in G to each transition.

For example, Figure 1 a) is a WFD-net, where $D = \{v_1, v_2, v_3\}$, $G = \{g_1(v_1), \neg g_1(v_1)\}$, $\text{Guard}(t_1) = g_1(v_1)$, $\text{Guard}(t_3) = \neg g_1(v_1)$, $Rd(t_1) = \{v_1\}$, $Wr(t_2) = \{v_2\}$ and $De(t_4) = \{v_1\}$. Moreover, $g_1 \otimes \neg g_1$ and $\text{Var}(g_1) = \text{Var}(\neg g_1) = \{v_1\}$.

As for a WFD-net, a state is generally called a *weak configuration*, and it includes a marking and the evaluations of data and guards.

Definition 3 (Weak configuration)**.** Let $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ be a WFD-net. $c = \langle m, \sigma, \eta \rangle$ is a weak configuration, where

1. m is a marking of (P, T, F);
2. $\sigma : D \to \{\top, \bot\}$ assigns a defined value ($\top$) or an undefined value ($\bot$) to each data element; and
3. $\eta : G \to \{\text{TRUE}, \text{FALSE}, \bot, \top\}$ assigns TRUE, FALSE, an undefined value ($\bot$) or a defined value ($\top$) to each guard.

In a weak configuration, (σ, η) represents a data state. Besides, we use the guards labeled with $*$ to represent their defined values ($\top$). As shown in Figure 1 b), $c_1 = \langle m_1, \sigma_1, \eta_1 \rangle = \langle [p_1 + p_2], \{v_1\}, \{*g_1\} \rangle$ is a weak configuration of the WFD-net in Figure 1 a), where $\sigma_1(v_1) = \top$ and $*g_1$ represents that $\eta_1(g_1) = \top$.

Given the definition of WFD-net, we discuss its *weak firing* rules of an enabled transition at a weak configuration.

Definition 4 (Weak enabling/firing rules)**.** Let $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ be a WFD-net. $t \in T$ is enabled at a weak configuration $c = \langle m, \sigma, \eta \rangle$, which is denoted by $c[t\rangle$, if

1. $m[t\rangle$;
2. $\forall d \in Rd(t) : \sigma(d) = \top$; and
3. $\forall d \in \text{Var}(\text{Guard}(t))$: $\sigma(d) \neq \bot$ and $\eta(\text{Guard}(t)) \in \{\text{TRUE}, \top\}$.

After firing an enabled transition t at c, a weak configuration $c' = \langle m', \sigma', \eta' \rangle$ is generated, where

1. $m[t\rangle m'$;
2. $\forall d \in De(t) : \sigma'(d) = \bot$;
3. $\forall d \in Wr(t) \setminus De(t) : \sigma'(d) = \top$;
4. $\forall d \in D \setminus (De(t) \cup Wr(t)) : \sigma'(d) = \sigma(d)$;
5. $\exists g \in \text{Guard}(t) : \text{Var}(g) \cap Wr(t) = \emptyset \Rightarrow \eta'(g) = \text{TRUE}$; and
6. $\forall g \in G, \forall d \in \text{Var}(g) : (\sigma'(d) = \top \Rightarrow \eta'(g)\top) \wedge ((g \notin \text{Guard}(t) \wedge \text{Var}(g) \cap Wr(t) = \emptyset) \Rightarrow \eta'(g) = \eta(g))$.

It is denoted as $c[t\rangle c'$.

For example, the transition t_0 in Figure 1 a) is enabled at the *initial weak configuration* c_0 and $c_0[t_0\rangle c_1$, where $c_0 = \langle [i], -, - \rangle$ and $c_1 = \langle [p_1 + p_2], \{v_1\}, \{*g_1\} \rangle$. After firing the transition t_0 and writing a new value into the data v_1, the evaluations of g_1 is not definite because v_1 is associated with this guard. We assign a defined value ($\top$) to this guard in c_1. Thus, firing t_0 generates one unique weak configuration in Figure 1 b).

According to the enabling and firing rule of transitions, the may-reachability of WFD-net is defined as follows.

Definition 5 (May-reachability [22]). Let $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ be a WFD-net. c_1 and c_2 are two configurations.

1. There is a may-step from c_1 to c_2, denoted by $c_1 \rightarrow_{may} c_2$, if there is a transition $t \in T$ and a set of configurations C such that: $c_1[t\rangle C \wedge c_2 \in C$.
2. c_2 is may-reachable from c_1 if there exists a sequence of configurations $c^{(1)}, \ldots,$ $c^{(n)}$ such that $c_1 \rightarrow_{may} c^{(1)} \rightarrow_{may} \cdots \rightarrow_{may} c^{(n)} \rightarrow_{may} c_2$. It is denoted as $c_1 \rightarrow^*_{may} c_2$.

The set of may-reachable configurations from c is denoted by $R(c)$. For example, there is a may-step from c_0 to c_1 in Figure 1 b), and the configuration $c_6 = \langle [p_3 + p_4], \{v_1, v_2\}, \{g_1\} \rangle$ is may-reachable from c_0.

2.3 Guard-Driven Reachability Graph

Although the classical reachability graph [22] is a fundamental method of analyzing and verifying a WFD-net, it easily suffers from the problems of state space explosion and pseudo configurations due to its guard evaluations and their exclusive relations [30]. Hence, we propose the *guard-driven reachability graph* (GRG) based on the weak configurations and the weak firing rules.

Definition 6 (Guard-driven reachability graph, GRG). Let $N = (P, T, F, D, G, Rd, Wr, De, \text{Guard})$ be a WFD-net and c_0 be its initial weak configuration. $GRG(N) = (V^+, E^+, \ell^+)$ is the guard-driven reachability graph of N, where

1. $V^+ = R(c_0)$;
2. $E^+ = \{(c, c') \mid c, c' \in R(c_0) \wedge \exists t \in T : c[t\rangle c'\}$; and
3. $\ell^+ : E^+ \rightarrow T \times G$ such that $\ell^+(c, c') = \langle t, \text{Guard}(t) \rangle$ if $(c, c') \in E^+$ and $c[t\rangle c'$.

For example, Figure 1 b) is the GRG of the WFD-net in Figure 1 a), where $c_0, c_1 \in V^+$, $e_0 = (c_0, c_1) \in E^+$ and $\ell^+(e_0) = \langle t_0, \text{TRUE} \rangle$.[2]

In the guard-driven reachability graph of a WFD-net, the guard as the condition of enabling a transition determines the unique successor state when firing the

[2] If $\text{Guard}(t_0) = \emptyset$, we use $\langle t_0, \text{TRUE} \rangle$ to represent this case.

transition. Therefore, the idea of guard-driven reachability graph is to show the execution of a WFD-net by the evaluations of guards.

3 DATA-FLOW ERRORS DETECTION METHOD BASED ON THE GRG OF WFD-NET

3.1 Data-Flow Errors

Data-flow errors are caused by improper data operations in workflow systems, which mainly include missing data, redundant data, lost data, and inconsistent data. We first define these data-flow errors in a WFD-net.

1. Missing Data

Missing data occurs when a business process of workflow systems is reading or deleting some data, but this data is not existing at this time.

Definition 7 (Missing Data)**.** A WFD-net N with an initial weak-configuration c_0 has an error of missing data if $\exists t \in T$, $\exists c \in R(c_0) : c = \langle m, \sigma, \eta \rangle \wedge \neg c[t \wedge m[t\rangle \wedge \exists d \in Rd(t) \cup De(t) : \sigma(d) = \perp$.

For example, the transition t_5 in Figure 1 is not enabled at the reachable weak-configuration $[p_3 + p_4; v_2; g_1]$ because it cannot read the data from v_3 since this data has never been written into any values. At this time, missing data occurs.

2. Inconsistent Data

The error of inconsistent data usually occurs in a concurrent workflow system when one business process is reading or writing or deleting some data, but another process is concurrently writing or deleting this data. Notice that two transitions t_1 and t_2 in a bounded WFD-net are *concurrent* at a weak-configuration $c = \langle m, \sigma, \eta \rangle$, if they satisfy that $c[t_1\rangle \wedge c[t_2\rangle \wedge ({}^\bullet t_1 \cap {}^\bullet t_2 = \emptyset \vee \forall p \in {}^\bullet t_1 \cap {}^\bullet t_2 : m(p) \geq 2)$ [29]. This is denoted by $t_1 \|_c t_2$.

Definition 8 (Inconsistent Data)**.** The error of inconsistent data takes place in a WFD-net if two concurrent transitions t_1 and t_2 satisfy $\exists c \in R(c_0) : (Rd(t_1) \cup Wr(t_1) \cup De(t_1)) \cap (Wr(t_2) \cup De(t_2)) \neq \emptyset$.

For example, two transitions t_2 and t_4 in Figure 1 are concurrently writing into the data v_2 at the reachable weak-configuration $[p_1 + p_2; v_1; *g_1]$. At this time, inconsistent data occurs.

3. Redundant Data

Redundant data occurs if a data is never read before it is deleted or the business process terminates.

Definition 9 (Redundant Data)**.** Σ has an error of redundant data if one of the following two conditions holds:

1. $\exists c_1, c_2 \in R(c_0)$, $\exists t_1 \in T$, $\exists v \in D : c_1[t_1\rangle c_2 \wedge v \in Wr(t_1) \wedge (\forall c_3 \in R(c_2), \forall t_2 \in T : c_3[t_2\rangle \rightarrow v \notin Rd(t_2))$;
2. $\exists c_1, c_2 \in R(c_0)$, $\exists t_1, t_2 \in T$, $\exists \sigma \in T^*$, $\exists v \in D : c_1[t_1\sigma\rangle c_2[t_2\rangle \wedge v \in Wr(t_1) \wedge v \in De(t_2) \wedge (\forall t_3 \in \sigma : v \notin Rd(t_3))$.

For example, the transition t_2 in Figure 1 is to overwrite the data v_2 at the reachable weak-configuration $[p_1 + p_2; v_1; *g_1]$. But the data has never been read until the business process terminates. Therefore, there is an error of redundant data.

4. Lost Data

Lost data means that once a data element is written into a value by a transition, it will never be read before it is written again by some follow-up transitions. In other word, the first value of this data element cannot be referenced again by other activities.

Definition 10 (Lost Data)**.** Σ has an error of lost data if $\exists c_1, c_2 \in R(c_0)$, $\exists t_1, t_2 \in T$, $\exists \sigma \in T^*$, $\exists v \in V : c_1[t_1\sigma\rangle c_2[t_2\rangle \wedge v \in Wr(t_1) \cap Wr(t_2) \wedge (\forall t_3 \in \sigma : v \notin Rd(t_3)$.

For example, the transition t_0 in Figure 1 writes a data value into v_1 at the initial weak-configuration. But this data is never be read before it is overwritten by t_3. Therefore, this is an error of lost data.

3.2 The Algorithm for Checking Data-Flow Errors Based on GRG

In order to check the above data-flow errors in a workflow system, we propose an algorithm based on GRG, which is shown in Algorithm 1.

- According to the definition of missing data, we can easily check this data-flow error by traversing each weak-configuration.
- At a weak-configuration, if two concurrent transitions are concurrently conducting some data operations on a data element, we can find out an error of inconsistent data.
- If an enabled transition is to write some value into a data element at a weak-configuration, we can traverse all successors of this weak-configuration. Then, the weak-configurations related to the operations on this data are obtained by the function

$$\text{FindRWDConfigs}(v, c', GRG(\Sigma), CT).$$

That is, we traverse all weak-configurations reachable from c', and obtain three reachable weak-configuration sets c_r, c_w and c_d, where c_r (resp. c_w, c_d) is the set

```
Procedure_1 FindRWDConfigs($v$, $c'$, $GRG(\Sigma)$, $CT$)
1: if $c' \notin CT$ then
2:    $CT$.add($c'$);
3:    Get all edges from $c'$, i.e., $E_2 = \{(c', c'') \mid (c', c'') \in E\}$;
4:    if $E_2 \neq \emptyset$ then
5:       for each $(c', c'') \in E_2$ do
6:          if $c'[t'\rangle c''$ or $c'[_c t'\rangle c''$ then
7:             if $v \in Rd(t')$ then
8:                $c_r$.add($c'$);
9:             end if
10:            if $v \in Wr(t')$ then
11:               $c_w$.add($c'$);
12:            end if
13:            if $v \in De(t')$ then
14:               $c_d$.add($c'$);
15:            end if
16:            if $v \notin Rd(t') \cup Er(t') \cup De(t')$ then
17:               FindRWD($v$, $c''$, $GRG(\Sigma)$, $CT$);
18:            end if
19:         end if
20:      end for
21:   end if
22: end if
end Procedure_1
```

of reachable weak-configurations at which there is a read (resp. write, delete) operation. Finally, according to these weak-configurations, we determine whether there is an error of redundant data or lost data.

It is noted that GRG can effectively reduce the state space and avoid pseudo states since it fully considers the characters of guard functions. As a result, Algorithm 1 only needs to traverse a smaller state space to detect data-flow errors in comparison with the classical reachability graph in [22]. Moreover, it also prevents from some negative influence by pseudo states.

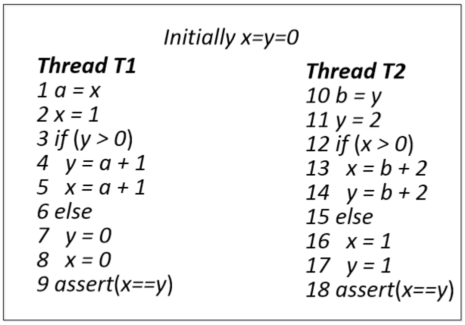
4 CASE STUDY

Our checking method for data-flow errors can be applied in the static program analysis. Figure 2 is a multi-thread program, and it is used to detect the errors of data inconsistency in the related work [9]. As our case study in this paper, we utilize it to check data-flow errors.

We first use a WFD-net to model this program, which is shown in Figure 3 a). Tables 1 and 2 list the related transitions and guards. Meanwhile, if we respec-

Algorithm 1 Data-flow error detection algorithm

```
Require: A WFD-net $N$
Ensure: All data-flow errors.
1: Initialize $C^\sharp = \emptyset$; /* The detected weak-configurations. */
2: Construct a GRG of $N$, i.e., $GRG(N) = (V^+, E^+, \ell^+)$.
3: for each $c \in R(c_0)$ such that $c \notin C^\sharp$ do
4:   $C^\sharp$.add($c$);
5:   if $\exists t \in T : \neg c[t \wedge m[t\rangle \wedge \exists d \in Rd(t) \cup De(t) : \sigma(d) = \perp$ then
6:     print Missing data;
7:   end if
8:   ----------------------------------
9:   if $\exists t_1, t_2 \in T : t_1 || t_2 \wedge (Rd(t_1) \cup Wr(t_1) \cup De(t_1)) \cap (Wr(t_2) \cup De(t_2)) \neq \emptyset$
     then
10:    print Inconsistent Data between $t_1$ and $t_2$;
11:  end if
12:  ----------------------------------
13:  Get all edges from $c$, i.e., $E_1 = \{(c, c') \mid (c, c') \in E^+\}$;
14:  for each $(c, c') \in E_1$ do
15:    if $c[t\rangle c'$ and $Wr(t) \neq \emptyset$ then
16:      for each $v \in Wr(t)$ do
17:        $CT = \emptyset$; /* The traversed weak-configurations */
18:        Set $c_r = c_w = c_d = \emptyset$;
19:        FindRWDConfigs($v, c', GRG(\Sigma), CT$);
           /* As shown in Procedure_1, it is used to compute $c_r$, $c_w$ and $c_d$ */
20:        if $c_w \neq \emptyset$ then
21:          print Lost Data;
22:        end if
23:        ----------------------------------
24:        if $c_d \neq \emptyset$ then
25:          print Redundant Data;
26:        end if
27:        ----------------------------------
28:        if $c_r = c_w = c_d = \emptyset$ then
29:          print Redundant Data;
30:        end if
31:        ----------------------------------
32:      end for
33:    end if
34:  end for
35: end for
```



```
                    Initially x=y=0
Thread T1                          Thread T2
1 a = x                            10 b = y
2 x = 1                            11 y = 2
3 if (y > 0)                       12 if (x > 0)
4   y = a + 1                      13   x = b + 2
5   x = a + 1                      14   y = b + 2
6 else                             15 else
7   y = 0                          16   x = 1
8   x = 0                          17   y = 1
9 assert(x==y)                     18 assert(x==y)
```

Figure 2. Pseudo-codes of a multi-thread program

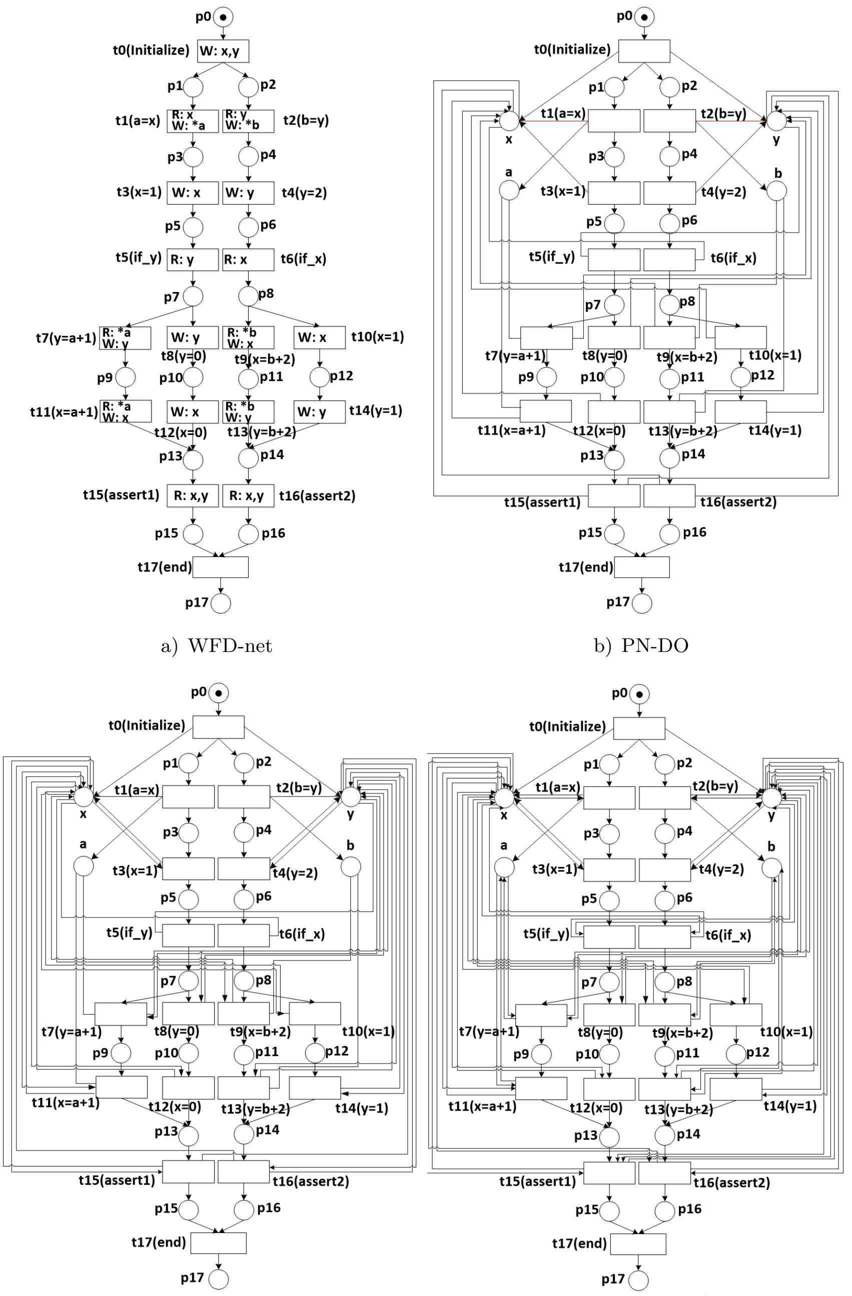
Transition ID	Codes	Read Data	Write Data
t_0	$x = y = 0$	–	x, y
t_1	$a = x$	x	a
t_2	$b = y$	y	b
t_3	$x = 1$	–	x
t_4	$y = 2$	–	y
t_5	if$(y > 0)$	y	–
t_6	if$(x > 0)$	x	–
t_7	$y = a + 1$	a	y
t_8	$y = 0$	–	y
t_9	$x = b + 2$	b	x
t_{10}	$x = 1$	–	x
t_{11}	$x = a + 1$	a	x
t_{12}	$x = 0$	–	x
t_{13}	$y = b + 2$	b	y
t_{14}	$y = 1$	–	y
t_{15}	assert1$(x == y)$	x, y	–
t_{16}	assert2$(x == y)$	x, y	–
t_{17}	end	–	–

Table 1. Program codes and data operations

tively use a PN-DO, a contextual net and a Petri net without read/write arcs to model this program, we can get a comparison between them, which are shown in Figure 3 and Table 3. It is clear that WFD-net needs a smaller space than others.

ID	Guard	ID	Guard
t_7	$g_1(y) : y > 0$	t_8	$\neg g_1(y) : y <= 0$
t_9	$g_2(x) : x > 0$	t_{10}	$\neg g_2(x) : x <= 0$

Table 2. Guards over transitions



a) WFD-net

b) PN-DO

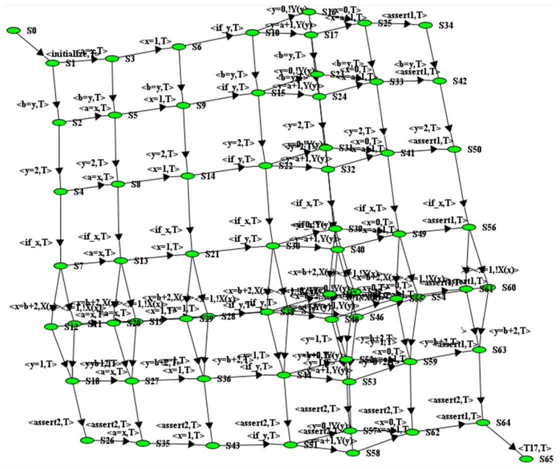
c) Contextual net

d) Petri net without read/write arcs

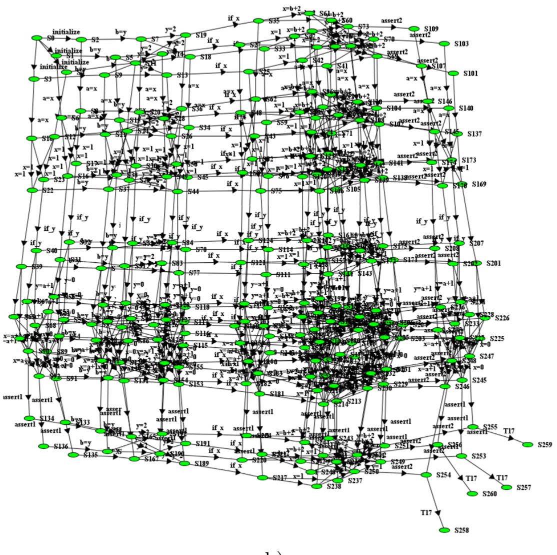
Figure 3. Different Petri nets for modeling the multi-thread program in Figure 2

	Nodes (Place & Transition)	Arcs
WFD-net [22, 26, 30]	36	38
PN-DO [31]	40	64
Contextual net [12]	40	74
Petri net without read/write arcs [2]	40	86

Table 3. The comparison between WFD-net and another Petri nets



a)



b)

Figure 4. a) GRG graph; b) the classical reachability graph

	(Weak) Configurations	Arcs	Time [ms]
RG	261	712	1 001
GRG	66	133	266

Table 4. The result comparison between RG and GRG

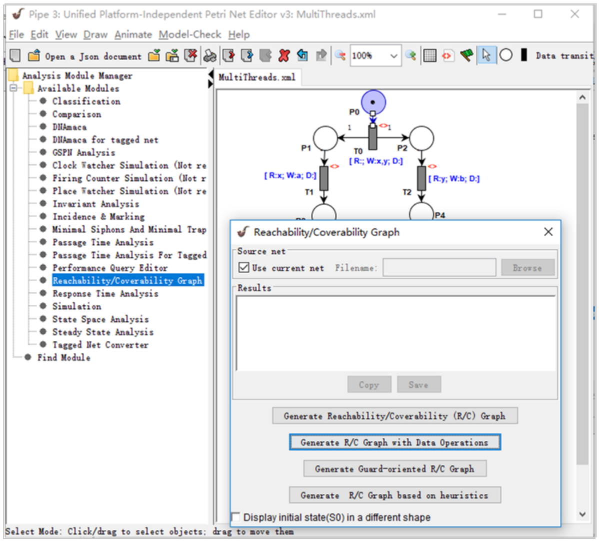

Pipe 3: Unified Platform-Independent Petri Net Editor v3: MultiThreads.xml
File Edit View Draw Animate Model-Check Help
Open a Json document
100%
Data transit
Analysis Module Manager
Available Modules
Classification
Comparison
DNAmaca
DNAmaca for tagged net
GSPN Analysis
Clock Watcher Simulation (Not re
Firing Counter Simulation (Not r
Place Watcher Simulation (Not re
Invariant Analysis
Incidence & Marking
Minimal Siphons And Minimal Trap
Passage Time Analysis
Passage Time Analysis For Tagged
Performance Query Editor
Reachability/Coverability Graph
Response Time Analysis
Simulation
State Space Analysis
Steady State Analysis
Tagged Net Converter
Find Module
MultiThreads.xml
P0
P1
P2
T0
[R:; W:x,y; D:]
[R:x; W:a; D:]
[R:y; W:b; D:]
T1
T2
P4
Reachability/Coverability Graph
Source net
Use current net
Filename:
Browse
Results
Copy
Save
Generate Reachability/Coverability (R/C) Graph
Generate R/C Graph with Data Operations
Generate Guard-oriented R/C Graph
Generate R/C Graph based on heuristics
Display initial state(S0) in a different shape
Select Mode: Click/drag to select objects; drag to move them

Figure 5. Our developed tool

As for the WFD-net in Figure 3 a), we utilize our tool [29] (see Figure 5) to construct its guard-driven reachability graph (GRG) and classical reachability graph (CRG), which are respectively shown in Figures 4 a) and 4 b). Table 4 gives a comparison between them in terms of state space and running time. Obviously, the former has an advantage over the latter.

Based on the GRG in Figure 4 a), we check the data-flow errors according to Algorithm 1. Table 5 lists these results. From these results, we can see that there exist some errors of inconsistent data because t_3 and t_9 (resp. t_{10}) are concurrent at the weak-configuration S_{13} and they satisfy $Wr(t_3) \cap Wr(t_9) = \{x\}$ (resp. $Wr(t_3) \cap Wr(t_{10}) = \{x\}$). The transitions t_4 and t_7 (resp. t_8) also suffer from the errors of inconsistent data. Moreover, t_{11} (resp. t_{12}) overwrites some data into x at weak-configurations S_1^+ after firing t_3, and t_{13} (resp. t_{14}) overwrites some data into y at weak-configurations S_2^+ after firing t_4. Therefore, there are some errors of lost data.

Data-Flow Errors	Weak-Configurations	Illustration
Missing Data	–	–
Inconsistent Data	S_{13}	t_3 and t_9 (or t_{10}) concurrently write some data into x
	S_{15}	t_4 and t_7 (or t_8) concurrently write some data into y
Redundant Data	–	–
Lost Data	$S_1^+ : \{S_{16}, S_{17}, S_{23}, S_{24}, S_{31}, S_{32}, S_{39}, S_{40}, S_{45}–S_{48}, S_{52}, S_{53}, S_{57}, S_{58}\}$	t_{11} (resp. t_{12}) overwrites some data into x
	$S_2^+ : \{S_{11}, S_{12}, S_{19}, S_{20}, S_{28}, S_{29}, S_{37}, S_{38}, S_{45}–S_{48}, S_{54}, S_{55}, S_{60}, S_{61}\}$	t_{13} (resp. t_{14}) overwrites some data into y

Table 5. The result of checking data-flow errors

5 EXPERIMENTS

We do a set of experiments in order to compare RG- and GRG-based methods for checking data-flow errors in terms of state space and runtime.

In our experiments, a tool is developed to generate RGs and GRGs of any bounded WFD-nets. It is based on PIPE (Platform Independent Petri Net Editor) [6], which is an open source tool of Petri net. Our tool can draw, edit, import and export a WFD-net.

Our experimental benchmarks are listed as follows:

- *KIT*[3] is a data set of BPMN 2.0 process models that describes 11 scenarios (including the business processes $BP_1 \sim BP_{11}$) with data specifications.
- *SystemC*, *blanc2010race* illustrates a SystemC (a modeling language) module.
- *AddGlobal*, *Sinha2010Staged* is an example of concurrency bugs when multi-threads access shared variables.

We utilize a PC with Intel Core i5-2400 CPU (3.10 GHz) and 4.0 GB memory to do experiments. We first use WFD-nets to model these benchmarks in our tool, and then respectively obtain their RGs and GRGs.

Based on GRGs, we can check data-flow errors. Table 6 presents the results of our experiments for all benchmarks. Obviously, the scale of GRG is much smaller than RG. Meanwhile, our GRG-based method spends less time to produce a GRG than the RG-based method. Naturally, the former has an advantage over the latter in terms of checking data-flow errors.

[3] `http://dbis.ipd.kit.edu/2134.php`, von2014detecting

Benchmark	RG			GRG			Errors
	Nos. of States	Nos. of Arcs	Time of RG	Nos. of States	Nos. of Arcs	Time of GRG	
BP_1	13	13	75.2	12	12	60.8	R
BP_2	17	18	70.6	16	16	65.8	R
BP_3	21	28	85.2	17	21	73.1	–
BP_4	15	14	70.3	15	14	69.4	–
BP_5	10	11	67.7	9	9	59.3	R
BP_6	23	43	73.3	18	30	62.7	R
BP_7	12	13	51.4	11	11	42.6	R
BP_8	103	103	318.5	29	28	71.8	–
BP_9	16	15	55.5	14	13	49.7	R
BP_{10}	36	40	73.3	30	32	66.0	–
BP_{11}	111	218	1 042.3	29	34	73.2	R
SystemC	33	62	76.6	25	39	62.5	I, L
AddGlobal	50	101	125.1	30	37	72.8	I, L

[1] Time: (ms).
[2] Errors: "I" denotes inconsistent data, "L" represents lost data, and "R" means redundant data.

Table 6. Experimental results

6 CONCLUSION

Petri net is widely used to check data-flow errors in workflow systems. As a special kind of Petri net, WFD-net is prominent in the modeling of control- and data-flows of business processes. Hence, we use a WFD-net to model workflow systems and its reachability graph to check data-flow errors in this paper. However, the classical reachability graphs of WFD-nets easily suffer from the problems of state space explosion and pseudo states. In order to avoid these problems, we propose a GRG-based method for checking data-flow errors. On one hand, our modeling method of WFD-net takes a smaller space than contextual net and PN-DO. On the other hand, our GRG-based method can effectively reduce the state space and avoid pseudo states in comparison with the classical reachability graph.

In the future work, we plan to do the following studies:

1. we utilize some existing techniques to reduce the scale of GRG, e.g., binary decision diagram (BDD) [5], abstraction [20] and partial order reduction [7]; and
2. we explore the unfolding-based technique of WFD-net [30] to check data-flow errors.

Acknowledgements

This paper was supported by Zhejiang Provincial Natural Science Foundation of China (Grant No. LQ20F020002), and in part by the Key Laboratory of Embedded

System and Service Computing (Ministry of Education) (Grant No. ESSCKF 2019-02).

REFERENCES

[1] van der Aalst, W. M. P.—van Hee, K. M.—ter Hofstede, A. H. M.—Sidorova, N.—Verbeek, H. M. W.—Voorhoeve, M.—Wynn, M. T.: Soundness of Workflow Nets: Classification, Decidability, and Analysis. Formal Aspects of Computing, Vol. 23, 2011, No. 3, pp. 333–363, doi: 10.1007/s00165-010-0161-4.

[2] Awad, A.—Decker, G.—Lohmann, N.: Diagnosing and Repairing Data Anomalies in Process Models. In: Rinderle-Ma, S., Sadiq, S., Leymann, F. (Eds.): Business Process Management Workshops (BPM 2009). Springer, Berlin, Heidelberg, Lecture Notes in Business Information Processing, Vol. 43, 2009, pp. 5–16, doi: 10.1007/978-3-642-12186-9_2.

[3] Baldan, P.—Bruni, A.—Corradini, A.—König, B.—Rodríguez, C.—Schwoon, S.: Efficient Unfolding of Contextual Petri Nets. Theoretical Computer Science, Vol. 449, 2012, pp. 2–22, doi: 10.1016/j.tcs.2012.04.046.

[4] Blanc, N.—Kroening, D.: Race Analysis for SystemC Using Model Checking. ACM Transactions on Design Automation of Electronic Systems (TODAES), Vol. 15, 2010, No. 3, Art. No. 21, doi: 10.1145/1754405.1754406.

[5] Butler, J. T.—Sasao, T.—Matsuura, M.: Average Path Length of Binary Decision Diagrams. IEEE Transactions on Computers, Vol. 54, 2005, No. 9, pp. 1041–1053, doi: 10.1109/TC.2005.137.

[6] Dingle, N. J.—Knottenbelt, W. J.—Suto, T.: PIPE2: A Tool for the Performance Evaluation of Generalised Stochastic Petri Nets. ACM SIGMETRICS Performance Evaluation Review, Vol. 36, 2009, No. 4, pp. 34–39, doi: 10.1145/1530873.1530881.

[7] Bokor, P.—Kinder, J.—Serafini, M.—Suri, N.: Supporting Domain-Specific State Space Reductions Through Local Partial-Order Reduction. 2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011). IEEE Computer Society, 2011, pp. 113–122, doi: 10.1109/ASE.2011.6100044.

[8] Guo, X.—Sun, S. X.—Vogel, D.: A Dataflow Perspective for Business Process Integration. ACM Transactions on Management Information Systems, Vol. 5, 2014, No. 4, Art. No. 22, 33 pp., doi: 10.1145/2629450.

[9] Huang, J.—Zhang, C.—Dolby, J.: CLAP: Recording Local Executions to Reproduce Concurrency Failures. ACM SIGPLAN Notices, Vol. 48, 2013, No. 6, pp. 141–152, doi: 10.1145/2499370.2462167.

[10] Jensen, K.—Kristensen, L. M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Science and Business Media, 2009, doi: 10.1007/b95112.

[11] Jiang, F.-C.—Hsu, C.-H.—Wang, S.: Logistic Support Architecture with Petri Net Design in Cloud Environment for Services and Profit Optimization. IEEE Transactions on Services Computing, Vol. 10, 2017, No. 6, pp. 879–888, doi: 10.1109/TSC.2016.2514506.

[12] Kähkönen, K.—Heljanko, K.: Testing Programs with Contextual Unfoldings. ACM Transactions on Embedded Computing Systems (TECS), Vol. 17, 2018, No. 1, Art. No. 23, doi: 10.1145/2810000.

[13] Kähkönen, K.—Saarikivi, O.—Heljanko, K.: Unfolding Based Automated Testing of Multithreaded Programs. Automated Software Engineering, Vol. 22, 2015, No. 4, pp. 475–515, doi: 10.1007/s10515-014-0150-6.

[14] Luan, W.—Qi, L.—Zhao, Z.—Liu, J.—Du, Y.: Logic Petri Net Synthesis for Cooperative Systems. IEEE Access, Vol. 7, 2019, pp. 161937-161948, doi: 10.1109/ACCESS.2019.2950971.

[15] Meda, H. S.—Sen, A. K.—Bagchi, A.: On Detecting Data Flow Errors in Workflows. ACM Journal of Data and Information Quality, Vol. 2, 2010, No. 1, Art. No. 4, 31 pp., doi: 10.1145/1805286.1805290.

[16] Montanari, U.—Rossi, F.: Contextual Nets. Acta Informatica, Vol. 32, 1995, No. 6, pp. 545–596, doi: 10.1007/BF01178907.

[17] Moutinho, F.—Gomes, L.: Asynchronous-Channels within Petri Net-Based GALS Distributed Embedded Systems Modeling. IEEE Transactions on Industrial Informatics, Vol. 10, 2014, No. 4, pp. 2024–2033, doi: 10.1109/TII.2014.2341933.

[18] Pradhan, A.—Joshi, R. K.: A Taxonomy of Consistency Models in Dynamic Migration of Business Processes. IEEE Transactions on Services Computing, Vol. 11, 2018, No. 3, pp. 562–579, doi: 10.1109/TSC.2017.2735413.

[19] Sadiq, S.—Orlowska, M.—Sadiq, W.—Foulger, C.: Data Flow and Validation in Workflow Modelling. Proceedings of the 15th Australasian Database Conference (ADC '04), Vol. 27, Australian Computer Society, Inc., 2004, pp. 207–214.

[20] Schlich, B.: Model Checking of Software for Microcontrollers. ACM Transactions on Embedded Computing Systems, Vol. 9, 2010, No. 4, Art. No. 36, 27 pp., doi: 10.1145/1721695.1721702.

[21] Sharma, D.—Pinjala, S.—Sen, A. K.: Correction of Data-Flow Errors in Workflows. Proceedings of the 25th Australasian Conference on Information Systems (ACIS), 2014, 10 pp.

[22] Sidorova, N.—Stahl, C.—Trčka, N.: Soundness Verification for Conceptual Workflow Nets with Data: Early Detection of Errors with the Most Precision Possible. Information Systems, Vol. 36, 2011, No. 7, pp. 1026–1043, doi: 10.1016/j.is.2011.04.004.

[23] Sinha, N.—Wang, C.: Staged Concurrent Program Analysis. Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE '10), 2010, pp. 47–56, doi: 10.1145/1882291.1882301.

[24] Sun, S. X.—Zhao, J. L.—Nunamaker, J. F.—Sheng, O. R. L.: Formulating the Data-Flow Perspective for Business Process Management. Information Systems Research, Vol. 17, 2006, No. 4, pp. 374–391, doi: 10.1287/isre.1060.0105.

[25] Trcka, N.—van der Aalst, W.—Sidorova, N.: Workflow Completion Patterns. 2009 IEEE International Conference on Automation Science and Engineering (CASE 2009), 2009, pp. 7–12, doi: 10.1109/COASE.2009.5234170.

[26] Trčka, N.—van der Aalst, W. M. P.—Sidorova, N.: Data-Flow Anti-Patterns: Discovering Data-Flow Errors in Workflows. In: van Eck, P., Gordijn, J.,

Wieringa, R. (Eds.): Advanced Information Systems Engineering (CAiSE 2009). Springer, Berlin, Heidelberg, Lecture Notes in Computer Science, Vol. 5565, 2009, pp. 425–439, doi: 10.1007/978-3-642-02144-2_34.

[27] Varea, M.—Al-Hashimi, B. M.—Cortés, L. A.—Eles, P.—Peng, Z.: Dual Flow Nets: Modeling the Control/Data-Flow Relation in Embedded Systems. ACM Transactions on Embedded Computing Systems, Vol. 5, 2006, No. 1, pp. 54–81, doi: 10.1145/1132357.1132360.

[28] Von Stackelberg, S.—Putze, S.—Mülle, J.—Böhm, K.: Detecting Data-Flow Errors in BPMN 2.0. Open Journal of Information Systems, Vol. 1, 2014, No. 2, pp. 1–19.

[29] Xiang, D.—Liu, G.—Yan, C.—Jiang, C.: Detecting Data Inconsistency Based on the Unfolding Technique of Petri Nets. IEEE Transactions on Industrial Informatics, Vol. 13, 2017, No. 6, pp. 2995–3005, doi: 10.1109/TII.2017.2698640.

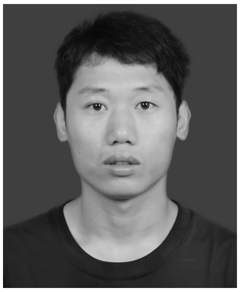
[30] Xiang, D.—Liu, G.—Yan, C.—Jiang, C.: A Guard-Driven Analysis Approach of Workflow Net with Data. IEEE Transactions on Services Computing, 2019, doi: 10.1109/TSC.2019.2899086.

[31] Xiang, D.—Liu, G.—Yan, C.—Jiang, C.: Detecting Data-Flow Errors Based on Petri Nets with Data Operations. IEEE/CAA Journal of Automatica Sinica, Vol. 5, 2018, No. 1, pp. 251–260, doi: 10.1109/JAS.2017.7510766.

[32] Yang, B.—Liu, G.—Xiang, D.—Yan, C.—Jiang, C.: A Heuristic Method of Detecting Data Inconsistency Based on Petri Nets. 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), 2018, pp. 202–208, doi: 10.1109/SMC.2018.00045.

[33] You, D.—Wang, S.—Seatzu, C.: Verification of Fault-Predictability in Labeled Petri Nets Using Predictor Graphs. IEEE Transactions on Automatic Control, Vol. 64, 2019, No. 10, pp. 4353–4360, doi: 10.1109/TAC.2019.2897272.

[34] Zou, J.—Liu, X.—Sun, H.—Zeng, J.: Live Instance Migration with Data Consistency in Composite Service Evolution. 2010 6th World Congress on Services, IEEE, 2010, pp. 653–656, doi: 10.1109/SERVICES.2010.76.



Dongming Xiang received his Ph.D. degree in computer science and technology from Tongji University, Shanghai, China, in 2018. He is currently Lecturer with the Department of Computer Science and Technology, Zhejiang Sci-Tech University. His research interests include model checking, Petri nets, business process management, and service computing.

Guanjun Liu received his Ph.D. degree in computer software and theory from Tongji University, Shanghai, China, in 2011. He was Post-Doctoral Research Fellow with the Singapore University of Technology and Design, Singapore, from 2011 to 2013. He was Post-Doctoral Research Fellow with the Humboldt University zu Berlin, Germany, from 2013 to 2014, supported by the Alexander von Humboldt Foundation. He is currently Professor with the Department of Computer Science and Technology, Tongji University. He has authored over 80 papers including 15 papers in IEEE/ACM Transactions and one book entitled Liveness of Petri Nets and Its Application (Tongji University Press, 2017). His research interests include Petri net theory, model checking, Web service, workflow, discrete event systems, and information security.