mathematics

Article

DICER 2.0: A New Model Checker for Data-Flow Errors of Concurrent Software Systems

Dongming Xiang [1],*, Fang Zhao [2] and Yaping Liu [3]

[1] The School of Information Science and Technology, Zhejiang Sci-Tech University, Hangzhou 310018, China
[2] The MoE Key Lab of Embedded System & Service Computing, Tongji University, Shanghai 201804, China; 1610471@tongji.edu.cn
[3] The School of Transportation Management, Zhejiang Institute of Communications, Hangzhou 310018, China; liuyp061054@zjvtit.edu.cn
* Correspondence: dmxiang@zstu.edu.cn

Abstract: Petri nets are widely used to model concurrent software systems. Currently, there are many different kinds of Petri net tools that can analyze system properties such as deadlocks, reachability and liveness. However, most tools are not suitable to analyze data-flow errors of concurrent systems because they do not formalize data information and lack efficient computing methods for analyzing data-flows. Especially when a concurrent system has so many concurrent data operations, these Petri net tools easily suffer from the state–space explosion problem and pseudo-states. To alleviate these problems, we develop a new model checker DICER 2.0. By using this tool, we can model the control-flows and data-flows of concurrent software systems. Moreover, the errors of data inconsistency can be detected based on the unfolding techniques, and some model-checking can be done via the guard-driven reachability graph (GRG). Furthermore, some case studies and experiments are done to show the effectiveness and advantage of our tool.

Keywords: petri net; concurrent software systems; model-checking; data-flows

Citation: Xiang, D.; Zhao, F.; Liu, Y. DICER 2.0: A New Model Checker for Data-Flow Errors of Concurrent Software Systems. *Mathematics* **2021**, *9*, 966. https://doi.org/10.3390/math9090966

Academic Editors: Tadashi Dohi and Vassilis C. Gerogiannis

Received: 11 March 2021
Accepted: 22 April 2021
Published: 25 April 2021

1. Introduction

Presently, concurrent software systems are widely used in our daily life. In particular, they are successfully applied in so many safety-critical scenarios, e.g., health-care, intelligent traffic, and stock exchange. Thus, how to guarantee the correctness of concurrent systems has become a bone of contention for people's lives and properties. In reality, the correctness of concurrent systems is closely related with control-flows and data-flows. However, the most existing studies mainly focus on the error detections of control-flows such as deadlocks, livelocks and compatibility [1–3]. In fact, concurrent systems are vulnerable to data-flow errors, e.g., missing data, lost data and data inconsistency [4–6]. Although the testing-based methods can detect these errors, they need to design a series of test cases to cover as many execution paths as possible. Due to the difficulty in the completeness of test cases, it is hard for these methods to guarantee a concurrent system error-free.

The Petri net-based model-checking is a prominent method/technique for analyzing data-flows of concurrent software systems. This is because Petri nets [7–10] have a great capability of explicitly specifying parallelism, concurrency and synchronization [11,12]. Thus, many different kinds of Petri nets are used to check data-flow errors, such as algebraic Petri net (or extended concurrent algebraic nets, ECANets), predicate/transitions net (PrTNet), and colored Petri nets (CPN), etc. Kheldoun et al. [13] transformed BPMN (Business Process Model and Notation) models of complex business processes into to Recursive ECATNets (RECATNets), which combine the expressive power of abstract data types with recursive Petri nets. Furthermore, they used rewriting logics to check proper terminations and LTL properties. Buchs et al. [14] proposed Concurrent Object-Oriented Petri

Nets (CO-OPN/2) to ensure the specifications of control/data-flows in a large distributed system. Barkaoui et al. [15] provided an approach for detecting data consistency with respect to a multilevel security policy based on ECATNets. He et al. [16] modeled smart contracts by predicate/transition nets, and then checked their correctness of pre/post-conditions. Wu et al. [17] developed a model-based method for quantitative safety analysis of safety-critical systems by Timed Colored Petri Nets (TCPNs). Yu et al. [18] proposed an E-commerce Business Process Net (EBPN) to verify the rationality and transaction consistency between trading parties. All these methods place emphasis on the formalizations of data structures and abstract data types. Thus, they are suitable to check data-flow errors caused by these aspects.

By comparison, some checking methods based on Petri nets focus on the modeling of conceptual data operations, e.g., *read*, *write* and *delete*. Dual Flow Nets (DFNs) [19] were proposed to model control- and data-flows of embedded systems. Awad et al. [20] mapped BPMN models into Petri nets, and then detected and repaired errors based on the work in [21]. Contextual net (C-net) [22,23] was proposed to model a concurrent read operation. Furthermore, its unfolding technique was developed to generate a minimal test suite for multi-threaded programs [24]. Referring to contextual nets, Petri Net with Data Operations (PN-DO) [5] was given to detect data-flow errors of concurrent software systems. However, these explicit formalizations of read/write arcs and data places easily increase the scales and complexity of Petri net models. Fortunately, *WFD-net* (WorkFlow net with Data) [4,25,26], as a high-level Petri net [8], is extended with conceptual labeling data operations and guards. Thus, on the one hand, a WFD-net can greatly model control-flows and data-flows of concurrent systems. On the other hand, the model scales of WFD-nets are much smaller than other Petri nets with data-flow arcs (e.g., read arcs, write arcs and delete arcs), such as C-net and PN-DO. Furthermore, WFD-net has been widely used to do model-checking, e.g., soundness [25], completion requirements [27] and data consistencies [28], although it is an easy way to model software systems. In general, these verification/analysis methods are based on the classical reachability graphs (CRG) [25] of WFD-nets. However, they easily suffer from the problems of state–space explosion and illegal states (or pseudo-states). This is because a state may have an exponential number of successor states since they are produced based on the possible values of all guards. Moreover, the exclusive logical relations (e.g., multiple choice conditions) between guards easily lead to pseudo-states. In order to alleviate these problems, we proposed a guard-driven reachability graph (GRG) of WFD-nets in our previous work [29].

Although a GRG of WFD-nets can describe all running information of concurrent systems and save their state–space compared with CRG, it still likely suffers from the state–space explosion problem. As shown in Figure 1, it easily leads to a rapid increase of state–space with the increase of concurrent operations of WFD-nets. This is because the interleaving semantics of GRG is based on the partial orders of fired transitions, and it describes the behaviors of concurrent systems only by global states. Thus, a GRG-based analysis method needs to find out all precedence relations between activities, and generates their successor states. Compared with the interleaving-semantics-based methods, some studies are conducted on a concurrency analysis of Petri nets [30,31]. In particular, the unfolding technique [32] can both alleviate the state space explosion problem and characterize the concurrency relations due to its true concurrency semantics [33]. Currently, this technique has been successfully applied in different kinds of model-checking, e.g., fault diagnosis [34], concurrent planning [35], test case generations [36], deadlock detection [37], and verifying soundness [38], reachability and coverability [39]. Thus, in view of these advantages, we proposed an unfolding-based method [5] to check errors of data inconsistency. Specifically, we use an acyclic net to represent all behaviors of a Petri net with data (PD-net [5]). On the one hand, all concurrent operations can be directly recorded in this acyclic net. On the other hand, this formal model can store all states and save much more space especially when a system has so many concurrent activities.

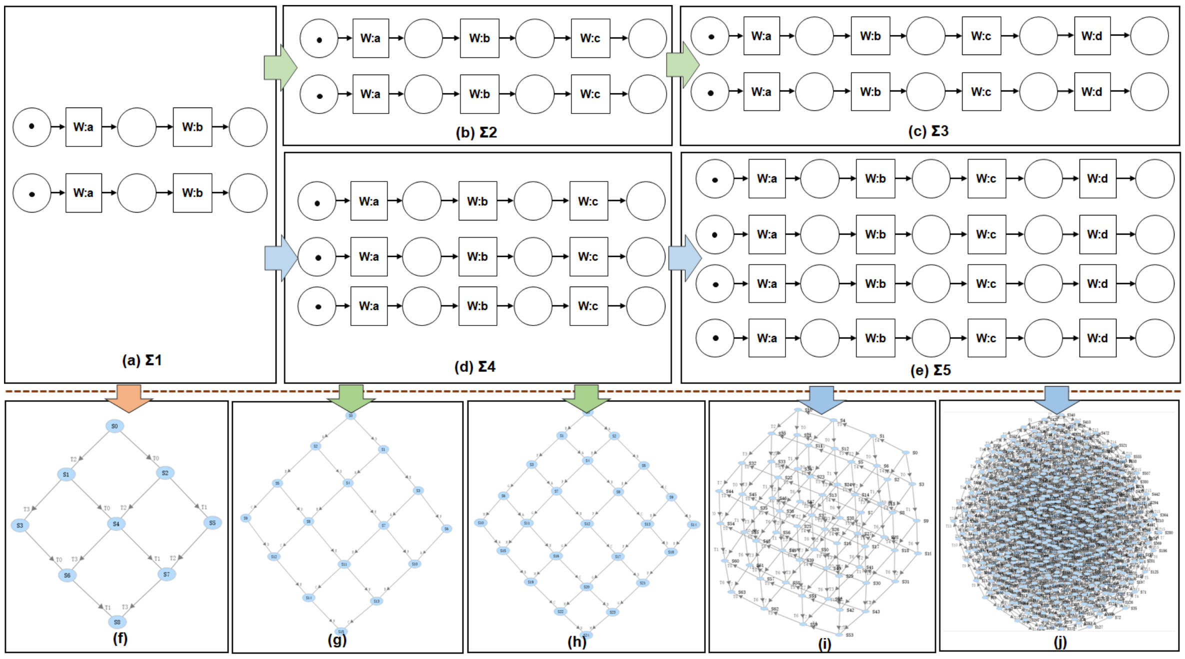


Figure 1. The state–space (reachability graphs) of WFD-nets and state–space explosion problems. (**f**) is the reachability graph of Σ_1 in (**a**); (**g**) is the reachability graph of Σ_2 in (**b**); (**h**) is the reachability graph of Σ_3 in (**c**); (**i**) is the reachability graph of Σ_4 in (**d**); and (**j**) is the reachability graph of Σ_5 in (**e**).

To support and improve the above previous work [5,29], we develop a new model checker DICER 2.0. Currently, there are many Petri net tools [40–42] such as PIPE, Snoopy, CPN Tools, Protos, and ProM. These tools can support different kinds of Petri net modeling, e.g., Place/Transition nets [7], Timed Petri nets [9], Stochastic Petri nets [10] and High-level Petri net [8]. Furthermore, they can be used to do structural analysis, generate condensed state spaces, construct reachability graphs, and analyze place/transition invariants. However, most of these tools fail in unfolding a Petri net. Although Mole, ERVunfold and Punf can do this work and conduct some model-checking (e.g., deadlocks, reachability and coverability), they cannot support the modeling and checking of data-flows that have been considered in some abstracted models, such as WFD-net [43] and PD-net [5]. Therefore, the most existing Petri net tools are not suitable to analyze data-flow errors of concurrent systems especially based on the unfolding techniques. The specified comparisons between some Petri net tools are summarized as Table 1.

In this paper, we develop DICER 2.0 to analyze data-flows of concurrent systems. Specifically, we can use this tool to model concurrent systems by general Petri nets, WFD-nets and PD-nets. Meanwhile, we can draw, edit, import and export these models in DICER 2.0. Moreover, the errors of data inconsistency can be detected based on the unfolding technique of PD-nets, and some GRG-based model-checking can be done in our tool.

This paper is organized as follows. Section 2 presents some basic notations. Section 3 introduces our model checker DICER 2.0. Section 4 gives two case studies on concurrent systems. Section 5 conducts a group of experiments to show the advantages of our tool. The last section concludes this paper.

Table 1. The comparison between some Petri net tools

Tools	Petri Nets	Functions	Branching Process	The Unfolding Techniques within Data-Flows	Data-Flow Error Detection
Snoopy		Graphical editor			
CPN Tools	P/T net	Reachability graph			
ProM	Timed Petri net	Condensed state spaces	×	×	×
PIPE2	High-level Petri net	P/T invariants			
PROTOS		Structural analysis			
Maude	ECATNet	Rewriting logic			
		LTL model-checking			
Acceleo+Maude	RECATNet	Transform RECATNets	×	×	✓
		into rewriting logics			
PIPE+	PrTNet	Modeling & simulating			
ERVunfold	P/T net	Deadlocks	✓	×	×
Tours		Test-case generation			
PUNF	Safe C-net	Reachability	✓	×	×
MOLE		Coverability			
	WFD-net	Detecting			
DICER 2.0	PD-net	data inconsistency	✓	✓	✓
	P/T net	Deadlocks			
	WF-net	Reachability			

2. Basic Notations

A *net* is a triple $N = (P, T, F)$, where P and T are two finite and disjoint sets, and they are called *place* and *transition*, respectively. $F \subseteq (P \times T) \cup (T \times P)$ denotes a *flow* relation. A *marking* of a net is a mapping function m: $P \rightarrow \mathbb{N}$, where $\mathbb{N}$ is a set of non-negative integers. In details, we use a multi-set to represent a marking. A net N with an initial marking m_0 is called a *Petri net* Σ [7] , i.e., $\Sigma = (N, m_0)$. Given a node $x \in P \cup T$, its *preset* and *postset* are respectively denoted by $^\bullet x$ and $x^\bullet$, where $^\bullet x = \{y \mid (y, x) \in F\}$ and $x^\bullet = \{y \mid (x, y) \in F\}$.

As a particular class of Petri net, *workflow net* (WF-net) is widely used to model control-flows of concurrent systems.

Definition 1. *A net $N = (P, T, F)$ is a WF-net (workflow net) [43,44] if*

(1) *there exists only one source place i and one sink place o satisfying $^\bullet i = \emptyset$ and $o^\bullet = \emptyset$; and*

(2) *each node $x \in P \cup T$ is on a path from i to o.*

Besides modeling control-flows of concurrent systems, we can use a net with some data information to formalize data-flows.

Definition 2. *A 7-tuple $N = (P, T, F, D, Read, Write, Delete)$ is a net with data (D-net) [5], if*

(1) *(P, T, F) is a net;*

(2) *D is a finite set of data elements;*

(3) *Read: $T \rightarrow 2^D$ is a labeling function of reading data;*

(4) *Write: $T \rightarrow 2^D$ is a labeling function of writing data; and*

(5) *Delete: $T \rightarrow 2^D$ is a labeling function of deleting data.*

Given two nodes $x, y \in P \cup T$ in an acyclic D-net $N = (P, T, F, D, Read, Write, Delete)$,

(1) x and y are in *causality* relation if the net N contains a path from x to y, which is denoted by $x \preceq y$. In particular, $x \prec y$ if $x \neq y$;

(2) x and y are in *conflict* relation if $\exists\, t_1, t_2 \in T$: $t_1 \preceq x$, $t_2 \preceq y$ and $^\bullet t_1 \cap {}^\bullet t_2 \neq \emptyset$, which is denoted by $x \# y$;

(3) x and y are in *backward-conflict* relation if $x^\bullet \cap y^\bullet \neq \emptyset$, which is denoted by $x \widetilde{\#} y$; or

(4) x and y are in *concurrency* relation if $\neg(x \prec y \vee y \prec x \vee x\#y)$, which is denoted by $x\ co\ y$, i.e., x and y are neither in causality relation nor in conflict relation.

OD-net (Occurrence Net with Data) is a simple acyclic net, which can be used in the unfolding technique of PD-nets [5].

Definition 3. *A D-net $N = (P, T, F, D, Read, Write, Delete)$ is an OD-net (Occurrence net with Data) [5] if*
(1) $\forall p \in P$: $|^\bullet p| \leq 1$;
(2) $\forall x, y \in P \cup T$: $x \prec y \Rightarrow y \not\prec x$; *and*
(3) *no transition is in self-conflict relation, i.e.,* $\forall t \in T$:$\neg(t\#t)$.

In an OD-net, places and transitions are called *conditions* and *events*, respectively. In general, we use $O = (B, E, G, D, Rd, Wr, De)$ to denote an occurrence net with data for convenience. With respect to this formalization, B, E and G are conditions, events and arcs, respectively. Rd, Wr and De are labeling functions of data operations (read, write and delete), respectively.

3. DICER 2.0

DICER 2.0 is developed to model and analyze the control-/data-flows of concurrent systems. It is the derivative version of our model checker for detecting data inconsistency [45]. Currently, we can use it to do many more model checking.

3.1. The Modeling of Concurrent Systems Based on the Petri Net with Data Information

As is well known, we usually use read/write arcs, data places, labeling functions of data operations and guards to formalize data-flows of concurrent systems [4,19,46]. In these formalizations, Petri nets such as DFN [19], PN-DO [47] and Awad method [20] mainly use data places and flow relations to model data operations, e.g., read, write and delete. Although these methods are suitable to accurately model the control structures of data-flows, it lacks formal semantic descriptions about shared reading and overwriting. Contextual net [46] can describe the concurrent (shared) reading operation by read arcs, but it needs extra data places and flow relations to formalize data-flows, and thus may be much more complex [48].

Compared with the above modeling methods, WFD-net [4,49] has a prominent advantage. It combines the traditional workflow nets with conceptual data operations, and uses labeling functions and guards to describe data operations and routing conditions, respectively. Thus, it is not only greatly suitable to model the control-flows and data-flows of a concurrent system but also much smaller than other Petri nets with data-operation arcs (e.g., contextual net and PN-DO) in the scales of nodes and arcs [48]. Now, this modeling method has been widely applied to various model-checking, e.g., detecting data-flow errors [4] and data inconsistency in the migrations of service cases [28], checking data inaccuracy [50] and completed requirements [27], and verifying may/must soundness of workflow systems [25].

Definition 4. *A workflow net with data (WFD-net) is a 9-tuple N =$(P, T, F, D, GD, Read, Write, Delete, Guard)$ [25], if*
(1) (P, T, F) *is a WF-net;*
(2) *D is a finite set of data elements;*
(3) *Read: $T \rightarrow 2^D$ is a labeling function of reading data;*
(4) *Write: $T \rightarrow 2^D$ is a labeling function of writing data;*
(5) *Delete: $T \rightarrow 2^D$ is a labeling function of deleting data;*
(6) *GD is a finite set of guards that are related with data elements in D; and*
(7) *Guard: $T \rightarrow GD$ is a labeling function of assigning guards to transitions.*

Referring to the labeling functions of data operations in WFD-nets, a *Petri net with data* (PD-net) [5] is proposed, i.e., a PD-net Σ is a D-net N with an initial marking m_0, i.e., $\Sigma = (N, m_0)$. Although this modeling method neglects the formalization of guards, it is much suitable for generating the unfolding of Peri nets with data information due to its simple structural semantics. For example, Σ is a WFD-net in Figure 2a, while Σ' is a PD-net in Figure 2c,d is its unfolding.

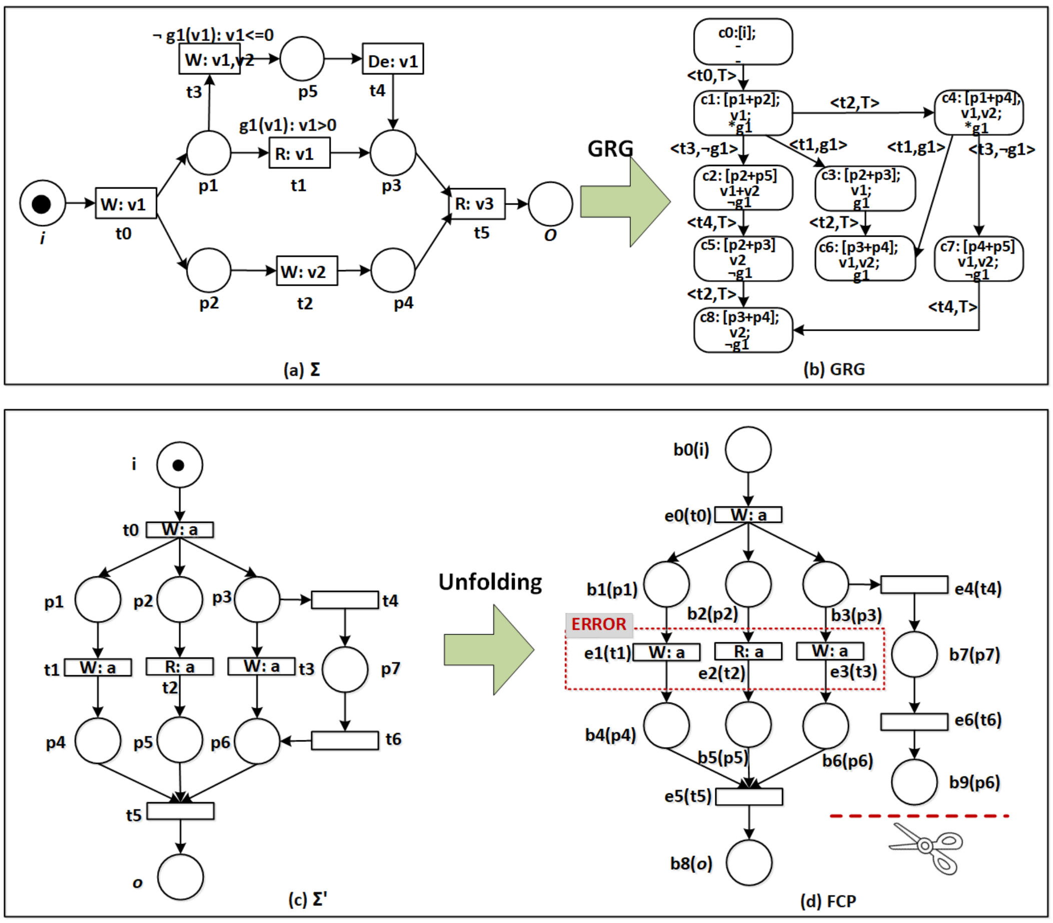


Figure 2. (**a**) A WFD-net Σ; (**b**) the guard-driven reachability graph (GRG) of Σ; (**c**) a PD-net Σ'; (**d**) the unfolding *FCP* of Σ'.

DICER 2.0 supports the modeling of WFD-nets and PD-nets. By this tool, we can formalize the control-/data-flows of concurrent systems. Furthermore, it provides a series of model-checking based on the guard-driven methods and unfolding techniques.

3.2. The Model-Checking Based on the GRG of WFD-Nets

The classical reachability graph [25] is a fundamental method for analyzing a WFD-net. However, this method easily suffers from the problems of state–space explosion and pseudo-states (or illegal states) due to its guard evaluations and their exclusive relations. Hence, we proposed a *Guard-driven Reachability Graph* (GRG) in our previous work [29], and now achieve this function in DICER 2.0.

To construct a GRG of WFD-nets, we define a state as a *weak configuration* in DICER 2.0, which includes a marking and some evaluations of data and guards.

Definition 5. *(Weak configuration) Given a WFD-net* $N = (P, T, F, D, GD, Read, Write, Delete, Guard)$, $c = \langle m, \sigma, \eta \rangle$ *is a weak configuration, if*

(1) (P, T, F) *is a WF-net, and* m *is its marking;*

(2) *a mapping function* $\sigma : D \to \{\top, \bot\}$ *assigns a defined value* $(\top)$ *or an undefined value* $(\bot)$ *to each data element; and*

(3) *a mapping function $\eta : GD \rightarrow \{TRUE, FALSE, \perp, \top\}$ assigns the values of TRUE , FALSE, $\perp$ or $\top$ to each guard.*

In DICER 2.0, we also define the basic enabling/firing rules of WFD-nets based on weak configurations.

Definition 6. *(Enabling/firing rules) Given a WFD-net N =(P, T, F, D, GD, Read, Write, Delete, Guard) and its weak configuration $c = \langle m, \sigma, \eta \rangle$, a transition t is enabled at c and denoted by $c[t\rangle$, if*

(1) $m[t\rangle$;

(2) $\forall v \in Read(t) : \sigma(v) = \top$; *and*

(3) $\forall v \in Varb(Guard(t))$: $\sigma(v) \neq \perp \wedge \eta(Guard(t)) \in \{TRUE, \top\}$, *where the function Varb is to obtain all variables in a guard.*

After firing a transition t at the weak configuration c, a new weak configuration $c' = \langle m', \sigma', \eta' \rangle$ can be generated, i.e., $c[t\rangle c'$, where

(1) $m[t\rangle m'$;

(2) $\forall v \in Write(t) \setminus De(t) : \sigma'(v) = \top$;

(3) $\forall v \in Delete(t) : \sigma'(v) = \perp$;

(4) $\forall v \in D \setminus (Write(t) \cup Delete(t)) : \sigma'(v) = \sigma(v)$;

(5) $\exists g \in Guard(t) : Write(t) \cap Var(g) = \emptyset \Rightarrow \eta'(g) = TRUE$; *and*

(6) $\forall g \in GD, \forall v \in Varb(g) : (\sigma'(v) = \top \Rightarrow \eta'(g) = \top) \wedge ((Write(t) \cap Varb(g) = \emptyset \wedge g \notin Guard(t)) \Rightarrow \eta'(g) = \eta(g))$.

Let c_1 and c_2 be two weak configurations of a WFD-net. c_2 is *may-reachable* from c_1, denoted as $c_1 \rightarrow^*_{may} c_2$, if there exist some weak configurations $c^{(1)}, c^{(2)}, \cdots, c^{(n)}$ such that $c_1[t_1\rangle c^{(1)}[t_2\rangle c^{(2)}[t_3\rangle \cdots c^{(n)}[t_3\rangle c_2$. Furthermore, a set of may-reachable weak configurations from c_1 is denoted by $R(c_1)$. Based on may-reachable sets and enabling/firing rules, we can formalize a GRG in DICER 2.0 as follows.

Definition 7. *Given a WFD-net N =(P, T, F, D, GD, Read, Write, Delete, Guard) and its initial weak configuration c_0, $GRG(N) = (V^+, E^+, \ell^+)$ is a guard-driven reachability graph (GRG), where*

(1) $V^+ = R(c_0)$, $E^+ = \{(c, c') \mid \exists c, c' \in R(c_0), \exists t \in T : c[t\rangle c'\}$, *and*

(2) ℓ^+: $E^+ \rightarrow T \times GD$ *such that* $(c, c') \in E^+ \wedge c[t\rangle c' \wedge \ell^+(c, c') = \langle t, Guard(t) \rangle$.

For example, Figure 2b shows a guard-driven reachability graph of Figure 2a, where g_1 and $\neg g_1$ are two exclusive guards, $c_0 = \langle [i], -, - \rangle$ and $c_1 = \langle [p_1 + p_2], \{v_1\}, \{*g_1\} \rangle$ are two weak configurations such that $c_0[t_0\rangle c_1$.

Since a GRG of a WFD-net contains all execution information of a concurrent system, we can traverse its reachable weak configurations by DICER 2.0 to do some model-checking such as deadlocks [51] and proper completeness [27], i.e., given a WFD-net N and its guard-driven reachability graph $GRG(N)$, o is its sink place and $c = \langle m, \sigma, \eta^+ \rangle$ is a weak configuration such that $c \in R(c_0)$.

- If $m(o) = 0$ and no transition is enabled at the weak configuration c, then c is a *deadlock*. Thus, we can check deadlocks in N according to this formal specification. For example, the WFD-net in Figure 2a have a deadlock at the weak configuration $c_8 : \langle [p_3 + p_4], \{v_2\}, \{\neg g_1\} \rangle$ because t_5 cannot read the data v_3 and no transition is enabled at this time.
- If $\forall c \in R(c_0) : m(o) > 0 \Rightarrow m = \{o\}$, then N is *properly completed*. For example, the WFD-net in Figure 2a is not properly completed since the final weak configuration is not reachable from the initial weak configuration and the sink place o has no token at this time.

3.3. The Model-Checking Based on the Unfolding Techniques of PD-Nets

Besides the model-checking based on GRGs of WFD-nets, DICER 2.0 can be used to detect errors of data inconsistency based on the unfolding techniques of PD-nets. We first define branching processes in DICER 2.0.

Definition 8. *Given a PD-net $\Sigma = (N, m_0) = (P, T, F, m_0, D, Read, Write, Delete)$ and an OD-net $O = (B, E, G, D, Rd, Wr, De)$, the mapping $h : B \cup E \to P \cup T$ is a homomorphism between Σ and O. (O, h) is a branching process if satisfying:*

(1) *$h(E) \subseteq T$ and $h(B) \subseteq P$;*

(2) *for each event e belonging to E, the restriction of h onto ${}^\bullet e$ (resp., $e^\bullet$) is a bijection between ${}^\bullet e$ and ${}^\bullet h(e)$ (resp., between $e^\bullet$ and $h(e)^\bullet$);*

(3) *the restriction of h onto $Min(O)$ is a bijection between $Min(O)$ and m_0;*

(4) *$\forall e_1, e_2 \in E$: $({}^\bullet e_1 = {}^\bullet e_2) \wedge (h(e_1) = h(e_2)) \Rightarrow e_1 = e_2$; and*

(5) *$\forall e \in E : Rd(e) = Read(h(e)) \wedge Wr(e) = Write(h(e)) \wedge De(e) = Delete(h(e))$.*

Given two branching processes $(O_i, h_i) = (B_i, E_i, G_i, D, Rd_i, Wr_i, De_i, h_i)$ and $i \in \{1, 2\}$, (O_1, h_1) is a prefix of (O_2, h_2) if $B_1 \subseteq B_2 \wedge E_1 \subseteq E_2$. All branching processes of a PD-net Σ forms a partial order set *w.r.t* the binary relation of *prefix*, and its greatest element is *Unfolding* [46], which is denoted by $Unf(\Sigma)$. Please note that the unfolding of a PD-net is also an occurrence net with data. Although the unfolding of a PD-net records its running information, it may be infinite if there exists an infinite execution path. Therefore, it needs to be truncated so as to get a *finite complete prefix* (FCP) [52]. In DICER 2.0, we refer to the ERV method [52] to cut off the unfolding of PD-nets, and then generate its FCP.

As a matter of fact, ERV method does not consider the Petri net modeling with data information. Moreover, it does not specify a highly efficient calculations on configurations, cuts and cut-off events. This is mainly caused by the following two facts. On the one hand, the most computing methods of configurations and cuts need a lot of repetitive calculations. On the other hand, once some new events are added into a given finite prefix, these methods usually match up them with all existing events and determine whether they are cut-off events or not. In order to solve these problems, DICER 2.0 uses recursion formulas and contextual information of events to compute configurations, concurrent conditions and cuts. Meanwhile, it uses backward conflicts to guide the calculations of cut-off events.

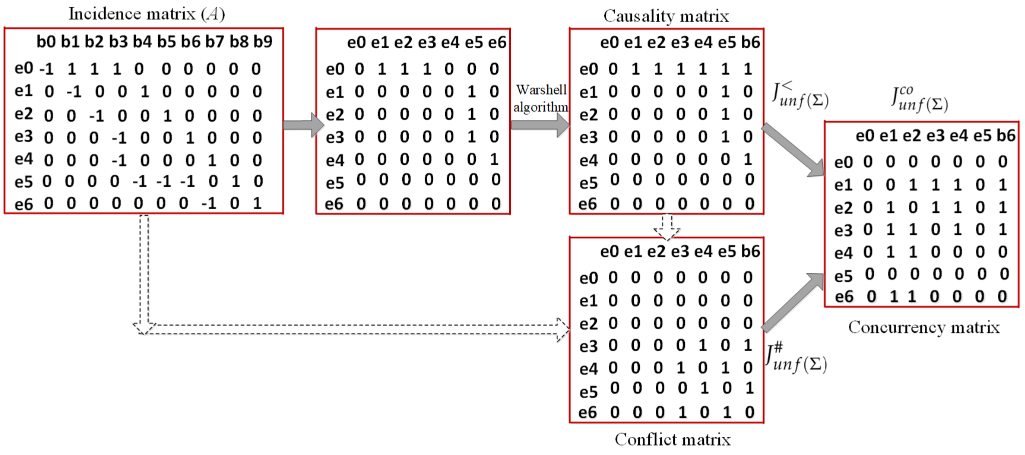
After generating an FCP of a PD-net Σ in DCIER 2.0, we can use its matrix manipulations to detect data inconsistencies since it contains the same behavioral information as the reachability graph of Σ (i.e., the completeness property [5] of FCP). In details, we first get an incidence matrix of this FCP, and then use Warshell algorithm to calculate its causality matrix $J^{\#}_{unf(\Sigma)}$. Afterwards, we obtain a conflict matrix $J^{\#}_{unf(\Sigma)}$ according to the mathematical definition of conflicts. Then, a concurrency matrix $J^{co}_{unf(\Sigma)}$ is calculated by $J^{<}_{unf(\Sigma)}$ and $J^{\#}_{unf(\Sigma)}$, *i.e.*, two events are in concurrency relation if they are neither in causality relation nor in conflict relation, i.e., $J^{\#}_{unf(\Sigma)} = [a_{(i,j)}]_{n\times n}$, $J^{co}_{unf(\Sigma)} = [a'_{(i,j)}]_{n\times n}$ and $J^{co}_{unf(\Sigma)} = [a''_{(i,j)}]_{n\times n}$, where $e_i, e_j \in E$ $(i, j \in \mathbb{N})$, and

$$a_{(i,j)} = \begin{cases} 1 & if\ e_i \# e_j \\ 0 & otherwise \end{cases} \qquad a'_{(i,j)} = \begin{cases} 1 & if\ e_i \# e_j \\ 0 & otherwise \end{cases} \qquad a''_{(i,j)} = \begin{cases} 1 & if\ e_i\ co\ e_j \\ 0 & otherwise \end{cases}$$

Based on the concurrency matrix $J^{co}_{unf(\Sigma)}$, we can check the errors of data inconsistency in Σ, i.e., there exists an error of data inconsistency if two concurrent events e_1 and e_2 have some data operations on a share data element, *i.e.*,

$$(Read(e_1) \cup Write(e_1) \cup Delete(e_1)) \cap (Write(e_2) \cup Delete(e_2)) \neq \varnothing.$$

For example, Figure 2d is an FCP of the PD-net in Figure 2c. Its related matrix calculations are conducted as shown in Figure 3. From this concurrency matrix, we can find that e_1, e_2 and e_3 are three concurrent events. Furthermore, they suffer from the errors of data inconsistency because $Write(e_1) \cap Read(e_2) \cap Write(e_3) \neq \emptyset$.



	b0	b1	b2	b3	b4	b5	b6	b7	b8	b9
e0	-1	1	1	1	0	0	0	0	0	0
e1	0	-1	0	0	1	0	0	0	0	0
e2	0	0	-1	0	0	1	0	0	0	0
e3	0	0	0	-1	0	0	1	0	0	0
e4	0	0	0	-1	0	0	0	1	0	0
e5	0	0	0	0	-1	-1	-1	0	1	0
e6	0	0	0	0	0	0	0	-1	0	1

	e0	e1	e2	e3	e4	e5	e6
e0	0	1	1	1	0	0	0
e1	0	0	0	0	0	1	0
e2	0	0	0	0	0	1	0
e3	0	0	0	0	0	1	0
e4	0	0	0	0	0	0	1
e5	0	0	0	0	0	0	0
e6	0	0	0	0	0	0	0

	e0	e1	e2	e3	e4	e5	b6
e0	0	1	1	1	1	1	1
e1	0	0	0	0	0	1	0
e2	0	0	0	0	0	1	0
e3	0	0	0	0	0	1	0
e4	0	0	0	0	0	0	1
e5	0	0	0	0	0	0	0
e6	0	0	0	0	0	0	0

	e0	e1	e2	e3	e4	e5	b6
e0	0	0	0	0	0	0	0
e1	0	0	0	0	0	0	0
e2	0	0	0	0	0	0	0
e3	0	0	0	0	1	0	1
e4	0	0	0	1	0	1	0
e5	0	0	0	0	1	0	1
e6	0	0	0	1	0	1	0

	e0	e1	e2	e3	e4	e5	b6
e0	0	0	0	0	0	0	0
e1	0	0	1	1	1	0	1
e2	0	1	0	1	1	0	1
e3	0	1	1	0	1	0	1
e4	0	1	1	0	0	0	0
e5	0	0	0	0	0	0	0
e6	0	1	1	0	0	0	0

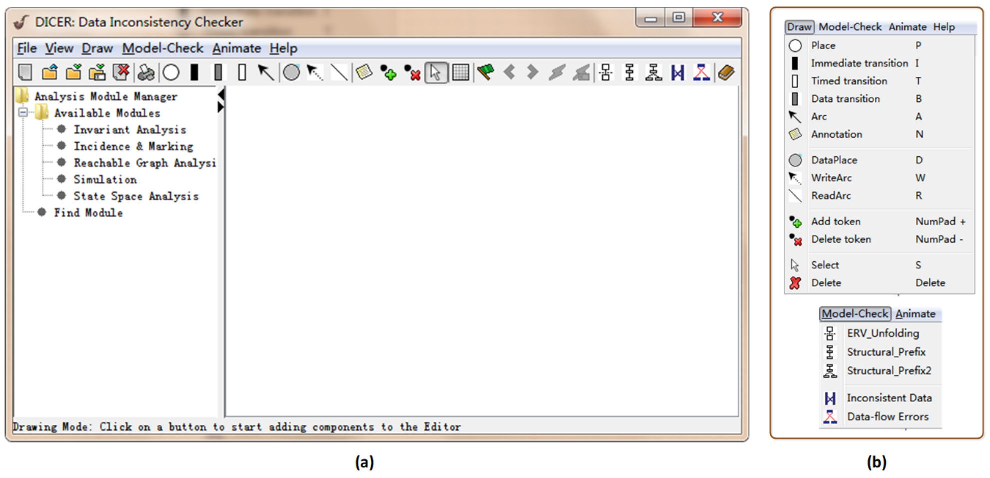
Figure 3. Some matrix manipulations on the FCP in Figure 2b.

3.4. The Implementations of DICER 2.0

Corresponding to the specified modeling and checking methods, we now introduce the basic framework and implementations of DICER 2.0.

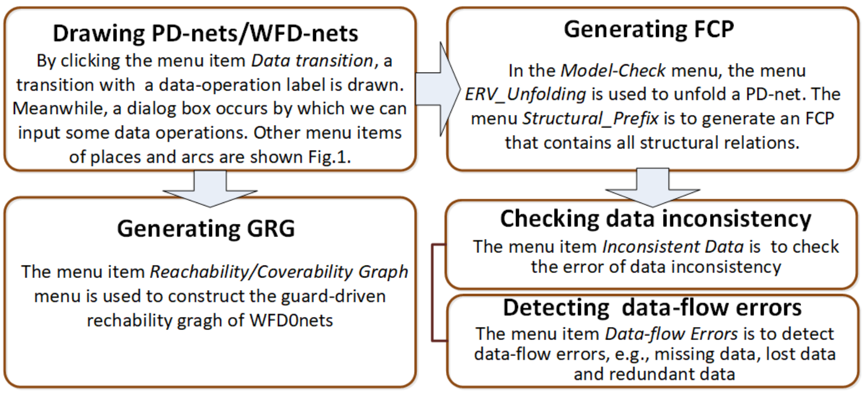
Figures 4 and 5 show the user interface (UI) and basic functions of DICER 2.0, respectively. Its framework is made up of two modules: graphical user interface (GUI) and model checker (MC), as shown in Figure 6. These two modules respectively correspond to the menus of drawing and model-checking in Figure 4.

- In the module of graphical user interface, Place/Transition nets, WFD-nets and PD-nets can be imported, exported, drawn and edited. The labeling functions of data operations (e.g., read, write and delete) can be added, deleted and modified in DICER 2.0. Moreover, different kinds of Petri nets are imported and exported in the format of an extended Petri Net Markup Language [53] (ePNML). In fact, ePNML provides a common interchange format for all types of Petri nets based on XML, and defines specifications of data operations and guard functions. As shown in Figure 7, the label $\langle isData \rangle$ formalizes data-flows of concurrent systems, including labeling functions of *read*, *write*, *delete* and *guards*. Since ePNML is an XML-based document, we can create or parse these Petri nets according to some configuration files, e.g., *GenerateObjectList*.xsl and *GeneratePNML*.xsl.
- In the module of model checker, Place/Transition nets and PD-nets can be unfolded, and then we can get their FCPs. As for the FCPs of PD-nets, we can use their matrix calculations (e.g., causality matrix, conflict matrix and concurrency matrix) to find out all concurrent events and then check errors of data inconsistency. Additionally, both classical reachability graphs and guard-driven reachability graphs of WFD-nets can be constructed in DICER 2.0. Furthermore, they are used to analyze some data-flow properties of concurrent systems, e.g., deadlocks, data inconsistency and soundness [29].



Drawing Mode: Click on a button to start adding components to the Editor

Figure 4. DICER 2.0 [45]. (**a**) Software interface; (**b**) the drawing menu and the model-checking menu.



Drawing PD-nets/WFD-nets
By clicking the menu item *Data transition*, a transition with a data-operation label is drawn. Meanwhile, a dialog box occurs by which we can input some data operations. Other menu items of places and arcs are shown Fig.1.

Generating FCP
In the *Model-Check* menu, the menu *ERV_Unfolding* is used to unfold a PD-net. The menu *Structural_Prefix* is to generate an FCP that contains all structural relations.

Generating GRG
The menu item *Reachability/Coverability Graph* menu is used to construct the guard-driven rechability gragh of WFD0nets

Checking data inconsistency
The menu item *Inconsistent Data* is to check the error of data inconsistency

Detecting data-flow errors
The menu item *Data-flow Errors* is to detect data-flow errors, e.g., missing data, lost data and redundant data

Figure 5. The basic functions of DICER 2.0.

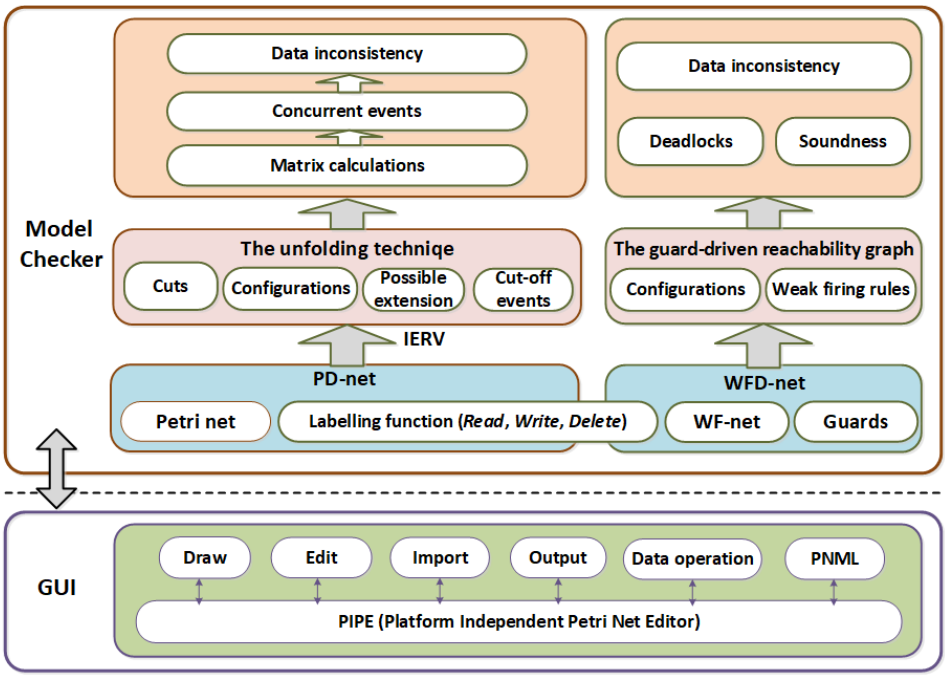
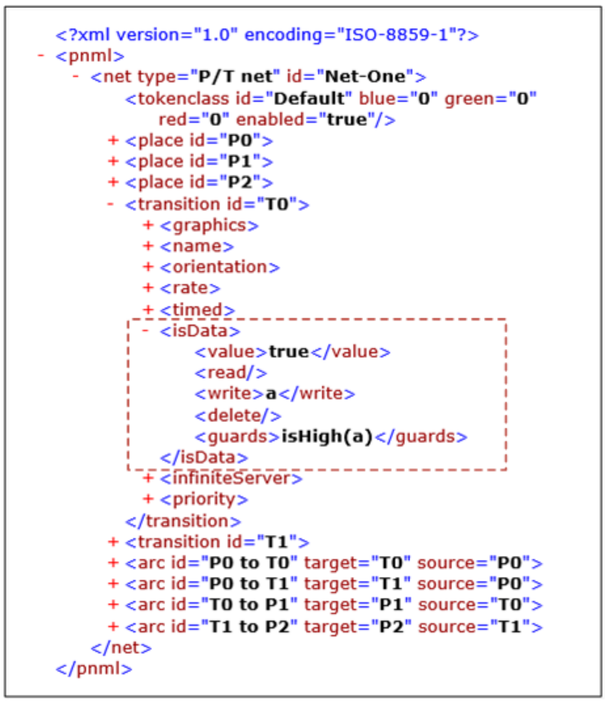


Figure 6. The basic framework of DICER 2.0.



```
  <?xml version="1.0" encoding="ISO-8859-1"?>
- <pnml>
  - <net type="P/T net" id="Net-One">
      <tokenclass id="Default" blue="0" green="0"
        red="0" enabled="true"/>
    + <place id="P0">
    + <place id="P1">
    + <place id="P2">
    - <transition id="T0">
      + <graphics>
      + <name>
      + <orientation>
      + <rate>
      + <timed>
      - <isData>
          <value>true</value>
          <read/>
          <write>a</write>
          <delete/>
          <guards>isHigh(a)</guards>
        </isData>
      + <infiniteServer>
      + <priority>
      </transition>
    + <transition id="T1">
    + <arc id="P0 to T0" target="T0" source="P0">
    + <arc id="P0 to T1" target="T1" source="P0">
    + <arc id="T0 to P1" target="P1" source="T0">
    + <arc id="T1 to P2" target="P2" source="T1">
    </net>
  </pnml>
```

Figure 7. An extended PNML [53] (ePNML) document of Petri nets with data operations and guards.

DICER 2.0 is developed-based on Platform Independent Petri Net Editor (PIPE) [40], which is an open source and graphical tool for drawing and analyzing Petri nets. In details, it is made up of a series of Java classes. Figure 8 shows the main hierarchy of these classes, which includes some flow information, inheritance relations, interfaces and methods.

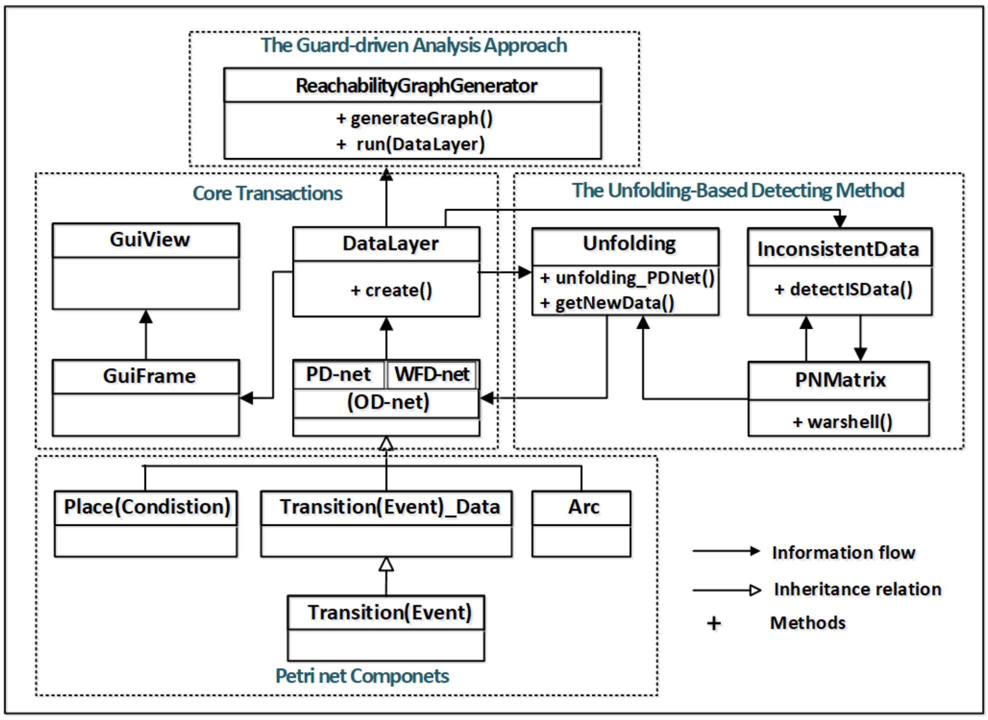


Figure 8. Main class hierarchy.

- The class *DataLayer* acts on the Petri net modeling of concurrent systems. It can be used to create, edit (e.g., add, move, or modify), import and export a PD-net or a

WFD-net. In this class, the method *getNewData*() is to obtain some information about the Petri net components of FCPs such as events, conditions and arcs.

- The class *Unfolding* is developed to unfold a PD-net or a Place/Transition net. Their FCPs can be generated by the method of *unfolding_PDNet*(*visual*, *"ERV"*, *null*). In this Java method, the parameter *visual* indicates whether an FCP needs to be displayed in the software interface, and the parameter *ERV* means a selected unfolding method, such as ERV, merged process, and directed unfolding.
- The class *ReachabilityGraphGenerator* is used to construct a guard-driven reachability graph of WFD-nets, and the methods *generateGraph*() and *run(DataLayer)* correspond to this function.
- The class *InconsistentData* is developed to check errors of data inconsistencies based on the unfolding of PD-nets, and the method *detectISData*() achieves this work in details.
- The classes *GuiView* and *GuiFrame* are used to create the front end, and display the software interface of DICER 2.0.
- A homomorphism from conditions to places (or from events to transitions) is represented by a hashmap. Its keys and values are in the form of $\langle Place, Place \rangle$ or $\langle Transition, Transition \rangle$, where *Place* and *Transition* are Java classes of Petri net components. Additionally, in order to improve the unfolding efficiency of PD-nets, we use some linked hash tables to store the contextual information of events and concurrent conditions, e.g., local configurations, pre/post-sets and cuts.

4. Case Study

To show the application scenarios of DICER 2.0, we give the following case studies.

4.1. Case _1: Intelligent Traffic Light System (ITIC)

Our first case study is conducted on an intelligent traffic light controller (ITIC) [54,55] for a North–South and East–West intersection. In this case study, the North–South (NS) is a main road, and the East–West (EW) is a rarely used country road. The North–South traffic light is always GREEN if the sensor of East–West Road is not activated. Otherwise, the North–South light will change from GREEN to YELLOW so as to give way to the East–West traffic. Additionally, some emergency vehicles can activate an emergency sensor. At this time, both the North–South and the East–West traffic lights need to turn RED.

In this case, of ITIC, we first use a WFD-net to model its business process, as shown in Figure 9. Table 2 shows all places and their meanings. The Boolean functions *select* $(EmgSensor, EWSensor)$ and $select(EmgSensor, EWSensor)$ are two exclusive guards on t_2 and t_3, respectively. By using DICER 2.0, we can draw and edit this WFD-net. Then, a guard-driven reachability graph is constructed, as shown in Figure 10. Based on this GRG, some properties can be verified by traversing each weak configuration (or state). For example, there is no deadlock in this ITIC system because there always exist enabled transitions at any weak configurations. Moreover, there is no error of data inconsistency since all concurrent transitions do not access a shared data element.

Table 2. Places and their meanings.

Place ID	Meanings
p_2	The yellow light of NS Road
p_3	The red light of NS Road
p_4	The green light of NS Road
p_6	The pre-green light of EW Road
p_7	The green light of EW Road
p_8	The yellow light of EW Road
p_{10}	The red light of EW Road
p_0, p_1, p_5, p_9	(*Control places*)

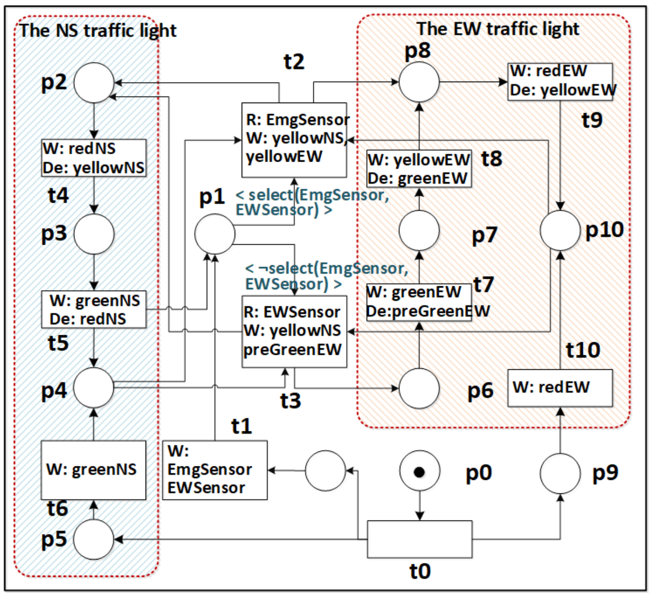
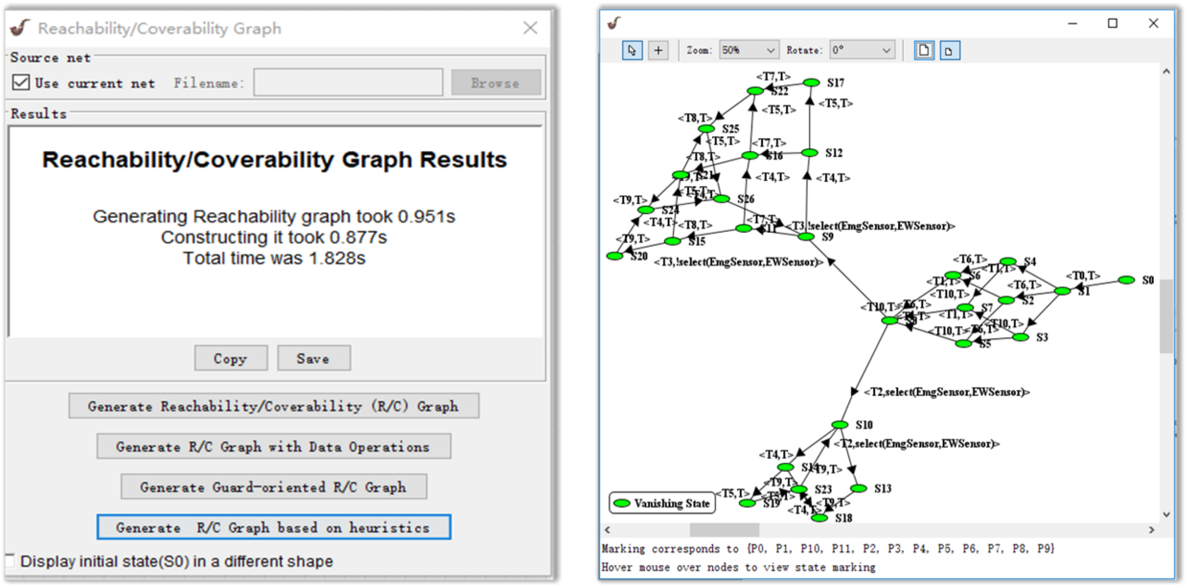


Figure 9. A WFD-net that models an intelligent traffic light system.


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
Zoom: 50%
Rotate: 0°
Vanishing State
<T2,select(EmgSensor,EWSensor)>
<T3,!select(EmgSensor,EWSensor)>
S0
S10
S13
S18

Reachability/Coverability Graph Results

Generating Reachability graph took 0.951s
Constructing it took 0.877s
Total time was 1.828s

```
Marking corresponds to {P0, P1, P10, P11, P2, P3, P4, P5, P6, P7, P8, P9}
Hover mouse over nodes to view state marking
```

Figure 10. A guard-driven reachability graph (GRG) of Figure 9. (**a**) A user interface for generating a GRG; (**b**) the visualization of a GRG.

4.2. Case _2: Health-Care Cyber-Physical System (HCPS)

The health-care cyber-physical system (HCPS) [56] consists of a series of devices such as e-health sensors, ambulance drones and ambulance vehicles. When an e-health sensor detects a cardiac arrest from patients, they will transmit this information to a controller, and then some warnings are sent to an emergency center. This center can also directly receive an emergency call from patients. After receiving these emergency messages, both drones and ambulances are ordered and sent to the emergency scene according to specific locations of patients.

In this case, of HCPS, we first use a PD-net to model its business process, as shown in Figure 11. Table 3 lists all transitions and their meanings. By using DICER 2.0, we can draw and edit this PD-net. Then, an FCP is generated, and some errors of data inconsistency are detected, which are respectively shown in Figure 12a,b. From Figure 12b, we can easily find that 12 concurrent events suffer from the errors of data inconsistency.

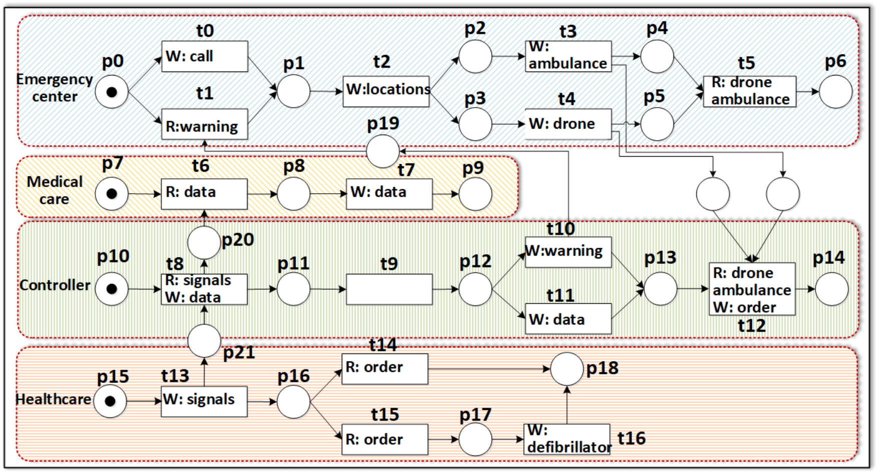


Figure 11. A PD-net that models a health-care cyber-physical system.

Table 3. Transitions and their meanings.

Transition ID	Meanings	Transition ID	Meanings
t_0	Receive emergency call	t_9	Control activity
t_1	Receive warning	t_{10}	Send warming
t_2	Find location	t_{11}	Store data
t_3	Send ambulance	t_{12}	Receive order
t_4	Send drone	t_{13}	Measure vital signals (E-health)
t_5	Supervise Drone	t_{14}	Movement of the ambulance
t_6	Receive data	t_{15}	Movement of the drone
t_7	Storage task	t_{16}	Install defibrillator
t_8	Send data		

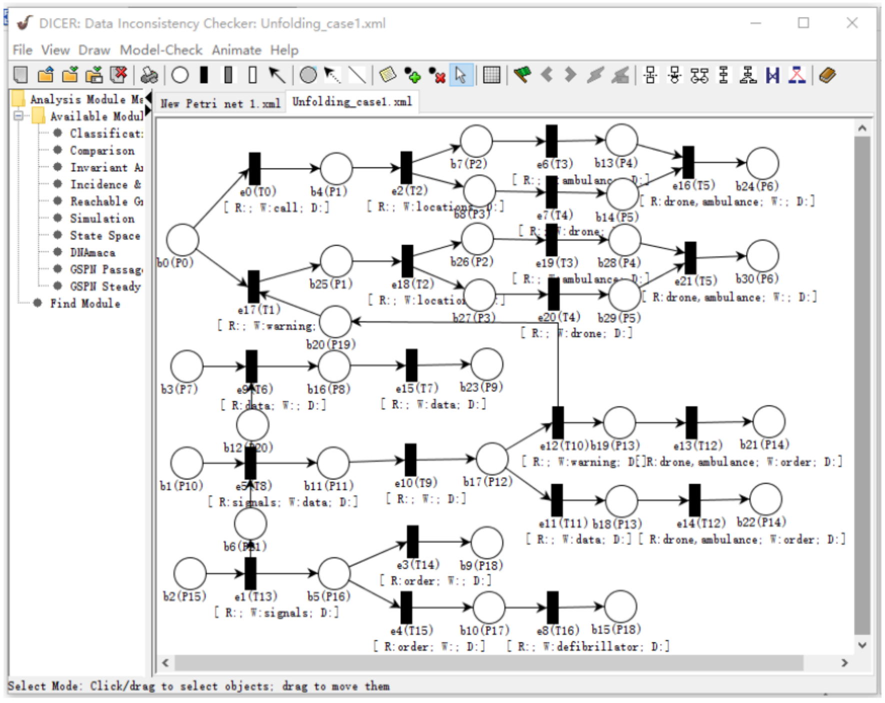
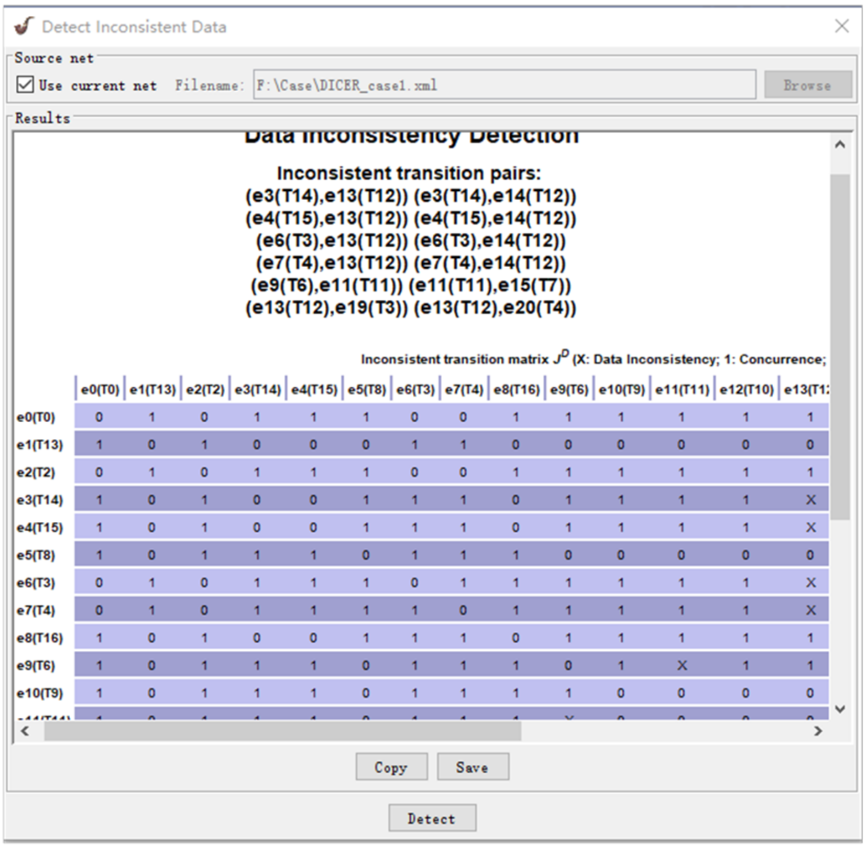


Figure 12. *Cont.*



F:\Case\DICER_case1.xml

Data Inconsistency Detection

Inconsistent transition pairs:
(e3(T14),e13(T12)) (e3(T14),e14(T12))
(e4(T15),e13(T12)) (e4(T15),e14(T12))
(e6(T3),e13(T12)) (e6(T3),e14(T12))
(e7(T4),e13(T12)) (e7(T4),e14(T12))
(e9(T6),e11(T11)) (e11(T11),e15(T7))
(e13(T12),e19(T3)) (e13(T12),e20(T4))

Inconsistent transition matrix J^D (X: Data Inconsistency; 1: Concurrence;

	e0(T0)	e1(T13)	e2(T2)	e3(T14)	e4(T15)	e5(T8)	e6(T3)	e7(T4)	e8(T16)	e9(T6)	e10(T9)	e11(T11)	e12(T10)	e13(T1
e0(T0)	0	1	0	1	1	1	0	0	1	1	1	1	1	1
e1(T13)	1	0	1	0	0	0	1	1	0	0	0	0	0	0
e2(T2)	0	1	0	1	1	1	0	0	1	1	1	1	1	1
e3(T14)	1	0	1	0	0	1	1	1	0	1	1	1	1	X
e4(T15)	1	0	1	0	0	1	1	1	0	1	1	1	1	X
e5(T8)	1	0	1	1	1	0	1	1	1	0	0	0	0	0
e6(T3)	0	1	0	1	1	1	0	1	1	1	1	1	1	X
e7(T4)	0	1	0	1	1	1	1	0	1	1	1	1	1	X
e8(T16)	1	0	1	0	0	1	1	1	0	1	1	1	1	1
e9(T6)	1	0	1	1	1	0	1	1	1	0	1	X	1	1
e10(T9)	1	0	1	1	1	0	1	1	1	1	0	0	0	0

(b)

Figure 12. Detecting errors of data inconsistency based on the unfolding techniques of PD-nets; (**a**) an FCP of the PD-net in Figure 11; (**b**) the detection results.

5. Experiments

5.1. Benchmarks

A group of experiments are done based on the following benchmarks to show the advantages of DICER 2.0. Please note that all of these experiments are implemented on a PC with 4.0G memory and Intel Core i5-2400 CPU.

- The *Index* program [57] is widely used for the experimental evaluation of multi-threads.
- The *Prime* benchmark (http://docs.oracle.com/cd/E19205-01/820-0619/gdvwv/index.html, accessed on 16 April 2021) is a tutorial program for detecting data race.
- The *Child_benefit* benchmark [58] is an example of transactional payment processes for child benefits.
- The *SystemC* benchmark [59] illustrates a SystemC (a modeling language) module.
- The *Driver* [60] benchmark describes a simplified model of bluetooth drivers.
- *AddGlobal* [61] gives an example of concurrency bugs.
- The *AppLoan* benchmark [62] describes a business process of approving property loan.
- The *Airport* benchmark [63] shows a business process of an airport check-in system.
- *Case_1* and *Case_2* are two case studies of intelligent traffic light system and health-care cyber-physical system, respectively.

5.2. Implementation and Results

(1) The experiments on the GRG of WFD-nets

The guard-driven reachability graph (GRG) of WFD-nets is an improved method for analyzing data-flows of concurrent systems. In this experiment, we use DICER 2.0 to compare it with the classical reachability graph (CRG) in terms of state–space and runtime.

We first use some WFD-nets to mode the benchmarks of *SystemC*, *AddGlobal*, *ApproveLoan*, *AirportCheck*, and *Driver* in DICER 2.0, and then respectively obtain their CRGs and GRGs. Table 4 shows the results of these experiments. Obviously, the scale of GRG is much smaller than RG. Meanwhile, our GRG-based method spends less time to produce a reachability graph than the CRG-based method.

Please note that although the GRGs of WFD-nets in Table 4 can save the state–space of concurrent systems compared with CRGs, they still likely suffer from the state–space explosion problem especially with the increase of concurrent (data) operations. In order to alleviate this problem, we conduct the following experiments based on the unfolding techniques.

(2) The experiments on the unfolding of PD-nets

The errors of data inconsistencies are usually detected based on reachability graphs (RGs). Thus, all states and arcs of RGs need to be traversed to do this work at worst. In this experiment, DICER 2.0 are used to detect these errors based on the unfolding techniques of PD-nets. In details, we compare their FCPs with RGs in terms of state–space, runtime and detection time.

We first use some PD-nets to model the benchmarks of *Child_benefit*, *Index* and *Prime* in DICER 2.0. Afterwards, their FCPs are generated, and some errors of data inconsistency are detected. Table 5 shows the scales (i.e., the numbers of nodes and arcs) of FCPs and RGs. Obviously, FCPs take up much smaller space than RGs. Meanwhile, this table also lists the time of generating FCPs and RGs. Thus, we can easily find that the former has a significant advantage over the latter.

Table 4. The experimental results of GRG and CRG in DICER 2.0.

Benchmarks	**CRG**			**GRG**		
	Nos. of States	**Nos. of Arcs**	**Time of Constructing CRGs**	**Nos. of States**	**Nos. of Arcs**	**Time of Constructing GRGs**
SystemC	33	62	76.6	25	39	62.5
AddGlob	50	101	125.1	30	37	72.8
AppLoan	51	112	149	17	22	63
Airport	15	16	320	12	13	220
Driver(2)	409	864	1987	172	283	532
Driver(4)	4117	14,696	14,863	2215	6094	6793
Driver(6)	22,921	105,988	95,333	13,754	48,346	45,461

CRG: Classical Reachability Graph; GRG: Guard-driven Reachability Graph. Time: (ms).

Table 5. The experimental results of unfolding PD-nets in DICER 2.0.

Benchmarks	**FCPs**					**RGs**		
	$\lvert E \cup B \rvert$	$\lvert G \rvert$	**Time of Unfolding**	**Time of Error Detection**	**Nos. of Errors**	**Nos. of States**	**Nos. of Arcs**	**Time of Constructing RGs**
Child_benefit	10	13	22	3	0	37	79	45
Index (5)	45	50	90	18	2	462	1680	557
Index (10)	90	100	180	44	3	7686	38,691	11,104
Index (15)	135	150	270	86	8	39,234	226,459	63,910
Index (20)	180	200	360	150	15	101,341	616,469	178,974
Prime (2)	37	39	75	13	0	82	197	102
Prime (4)	69	73	141	29	1	1369	5829	1795
Prime (6)	101	107	207	54	3	12,380	69,893	19,922
Prime (8)	133	141	273	92	7	75,538	509,004	160,541

Time: (ms).

(3) The comparison experiments between DICER 2.0 and other Petri net tools.

To further show the advantage of DICER 2.0, we make some comparisons between DICER 2.0 and other existing Petri net tools, e.g., PIPE, Tina and Punf. We select these tools based on the following considerations.

- The same or similar runtime environments.
- The same or similar functions and features.
- Available installations.

In these experiments, we first implement the benchmarks of *Case*_1 and *Case*_2 into different Petri net tools, and then we can get their experimental results. Tables 6 and 7 respectively show comparisons on the performance and functions of different Petri net tools. From these tables, we can find that DICER 2.0 supports the WFD-net modeling of concurrent systems, constructing GRGs, unfolding PD-nets and detecting errors of data inconsistency, while other Petri net tools do not. Please note that we must model data operations by data places and their related flows in Tina, PIPE and Punf because these tools cannot support the formalizations of labeling functions and guard functions. With respect to this modeling method, we can find that the model scales of *Case*_1 and *Case*_2 by these tools is much larger than WFD-net by DICER 2.0. Meanwhile, due to the lack of guard functions, these tools cannot model routing path conditions. Naturally, its reachability graph (by Tina and PIPE) is smaller than our GRG. Additionally, we cannot get an FCP of *Case*_2 by Punf because it cannot support the unfolding of unsafe Petri nets.

Table 6. The comparison experiments on the performance of DICER 2.0 and other Petri net tools.

Tools	**Case_1**			**Case_2**			
	Modeling ($\|P \cup T \cup F\|$)	**CRG**	**GRG**	**Modeling** ($\|P \cup T \cup F\|$)	**RG**	**FCP** ($\|B \cup E \cup G\|$)	**Detecting Data Inconsistency**
DICER 2.0	31	77	68	87	608	137	1.0 (*ms*)
PunF	87	–	–	125	–	–	–
Improved PIPE	31	77	–	87	608	–	–
Tina	87	53	–	125	608	–	–
PIPE	87	53	–	125	608	–	–

CRG: Classical Reachability Graph; GRG: Guard-driven Reachability Graph; RG: Reachability Graph. Data operations are modeled by data places and their related flows in Tina, PIPE and Punf because they cannot support the formalizations of labeling functions, guard functions and data-flow arcs.

Table 7. The comparison experiments on the functions of DICER 2.0 and other Petri net tools.

	Functions \ **Tools**	**DICER 2.0**	**Tina**	**PIPE**	**Punf**	**Improved PIPE**
Case_1	WFD-net	■	□	□	□	■
	Reachability graph	■	■	■	■	■
	Guard-driven reachability graph	■	□	□	□	□
	Unfolding	■	□	□	■	□
	Unfolding within data-flows	□	□	□	□	□
	Checking data inconsistency	■	□	□	□	□
Case_2	WFD-net	■	□	□	□	■
	Reachability graph	■	■	■	■	■
	Guard-driven reachability graph	■	□	□	□	□
	Unfolding	■	□	□	□	□
	Unfolding within data-flows	■	□	□	□	□
	Checking data inconsistency	■	□	□	□	□

6. Conclusions

Data-flow analysis plays an important role in the correctness verification of concurrent software systems. Petri net-based model checkings are a prominent method/technique for analyzing these data-flows. Currently, many different kinds of Petri nets have been used to do this work such as algebraic Petri net, predicate/transitions net, and colored Petri nets. WFD-net, as a high-level Petri net, is extended with conceptual labeling data operations. Thus, it can greatly model control/data- flows of concurrent systems. Moreover, its model scale is much smaller than other Petri nets with data-flow arcs such as C-net and PN-DO. Furthermore, WFD-net has been widely used to do model checkings. However, concurrent data operations and guard functions easily lead to the problems of state–space explosion and pseudo-states. In order to alleviate these problems, we proposed some efficient methods to detect data-flow errors and verify some properties. In this paper, we develop a new model checker DICER 2.0. By this tool, we can do a series of model checkings, e.g., detecting data inconsistencies based on the unfolding technique of PD-nets, and checking deadlocks via the GRG of WFD-nets.

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

(1) The unfolding methods of WFD-nets are studied to check many more data-flow errors and concurrency bugs [64,65] of concurrent systems;

(2) DICER 2.0 is further improved to support many more efficient model checkings; and

(3) Timed concurrent systems are modeled and checked by the unfolding techniques of Petri nets.

Author Contributions: D.X. proposed the idea in this paper and prepared the software application; D.X. and F.Z. designed the experiments; D.X. performed the experiments; Y.L. analyzed the data; D.X. wrote the paper; All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported in part by National Natural Science Foundation of China under Grant 62002328, Zhejiang Provincial Natural Science Foundation of China under Grant *LQ*20*F*020002, and in part by the Key Laboratory of Embedded System and Service Computing (Ministry of Education) under Grant *ESSCKF* 2019-02.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Not applicable.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Liu, G.; Jiang, C.; Zhou, M. Process nets with channels. *IEEE Trans. Syst. Man Cybern. Part A Syst. Hum.* **2012**, *42*, 213–225. [CrossRef]
2. You, D.; Wang, S.G.; Seatzu, C. Verification of Fault-predictability in Labeled Petri Nets Using Predictor Graphs. *IEEE Trans. Autom. Control* **2019**, *64*, 4353–4360. [CrossRef]
3. Li, W.; Xia, Y.; Zhou, M.; Sun, X.; Zhu, Q. Fluctuation-aware and predictive workflow scheduling in cost-effective Infrastructure-as-a-Service clouds. *IEEE Access* **2018**, *6*, 61488–61502. [CrossRef]
4. Trčka, N.; Van der Aalst, W.M.; Sidorova, N. Data-flow anti-patterns: Discovering data-flow errors in workflows. In *International Conference on Advanced Information Systems Engineering*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 425–439.
5. Xiang, D.; Liu, G.; Yan, C.; Jiang, C. Detecting data inconsistency based on the unfolding technique of petri nets. *IEEE Trans. Ind. Inform.* **2017**, *13*, 2995–3005. [CrossRef]
6. Liu, C.; Zeng, Q.; Duan, H.; Wang, L.; Tan, J.; Ren, C.; Yu, W. Petri net based data-flow error detection and correction strategy for business processes. *IEEE Access* **2020**, *8*, 43265–43276. [CrossRef]
7. Murata, T. Petri nets: Properties, analysis and applications. *Proc. IEEE* **1989**, *77*, 541–580. [CrossRef]
8. Gerogiannis, V.C.; Kameas, A.D.; Pintelas, P.E. Comparative study and categorization of high-level petri nets. *J. Syst. Softw.* **1998**, *43*, 133–160. [CrossRef]
9. Zuberek, W.M. Timed Petri nets definitions, properties, and applications. *Microelectron. Reliab.* **1991**, *31*, 627–644. [CrossRef]
10. Balbo, G. Introduction to generalized stochastic Petri nets. In *Proceedings of the 7th International Conference on Formal Methods for Performance Evaluation*; Springer: Berlin/Heidelberg, Germany, 2007; pp. 83–131.

11. Luan, W.; Qi, L.; Zhao, Z.; Liu, J.; Du, Y. Logic Petri Net Synthesis for Cooperative Systems. *IEEE Access* **2019**, *7*, 161937–161948. [CrossRef]
12. Moutinho, F.; Gomes, L. Asynchronous-channels within Petri net-based GALS distributed embedded systems modeling. *IEEE Trans. Ind. Inform.* **2014**, *10*, 2024–2033. [CrossRef]
13. Kheldoun, A.; Barkaoui, K.; Ioualalen, M. Formal verification of complex business processes based on high-level Petri nets. *Inf. Sci.* **2017**, *385*, 39–54. [CrossRef]
14. Buchs, D.; Guelfi, N. A formal specification framework for object-oriented distributed systems. *IEEE Trans. Softw. Eng.* **2000**, *26*, 635–652. [CrossRef]
15. Barkaoui, K.; Ayed, R.B.; Boucheneb, H.; Hicheur, A. Verification of workflow processes under multilevel security considerations. In Proceedings of the 2008 Third International Conference on Risks and Security of Internet and Systems, Tozeur, Tunisia, 28–30 October 2008; pp. 77–84.
16. He, X. Modeling and Analyzing Smart Contracts using Predicate Transition Nets. In Proceedings of the 2020 IEEE 20th International Conference on Software Quality, Reliability and Security Companion (QRS-C), Macau, China, 11–14 December 2020; pp. 108–115.
17. Wu, D.; Zheng, W. Formal model-based quantitative safety analysis using timed Coloured Petri Nets. *Reliab. Eng. Syst. Saf.* **2018**, *176*, 62–79. [CrossRef]
18. Yu, W.; Yan, C.; Ding, Z.; Jiang, C.; Zhou, M. Modeling and validating e-commerce business process based on Petri nets. *IEEE Trans. Syst. Man Cybern. Syst.* **2013**, *44*, 327–341. [CrossRef]
19. 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 Trans. Embed. Comput. Syst. (TECS)* **2006**, *5*, 54–81. [CrossRef]
20. Awad, A.; Decker, G.; Lohmann, N. Diagnosing and repairing data anomalies in process models. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2009; pp. 5–16.
21. Sharma, D.; Pinjala, S.; Sen, A.K. Correction of Data-flow Errors in Workflows. In Proceedings of the 25th Australasian Conference on Information Systems (ACIS), Auckland, New Zealand, 8–10 December 2014.
22. Baldan, P.; Bruni, A.; Corradini, A.; König, B.; Rodríguez, C.; Schwoon, S. Efficient unfolding of contextual Petri nets. *Theor. Comput. Sci.* **2012**, *449*, 2–22. [CrossRef]
23. Montanari, U.; Rossi, F. Contextual nets. *Acta Inform.* **1995**, *32*, 545–596. [CrossRef]
24. Kähkönen, K.; Heljanko, K. Testing Programs with Contextual Unfoldings. *ACM Trans. Embed. Comput. Syst. (TECS)* **2017**, *17*, 1–25. [CrossRef]
25. Sidorova, N.; Stahl, C.; Trčka, N. Soundness verification for conceptual workflow nets with data: Early detection of errors with the most precision possible. *Inf. Syst.* **2011**, *36*, 1026–1043. [CrossRef]
26. Yang, B.; Liu, G.; Xiang, D.; Yan, C.; Jiang, C. A Heuristic Method of Detecting Data Inconsistency Based on Petri Nets. In Proceedings of the 2018 IEEE International Conference on Systems, Man, and Cybernetics (SMC), Miyazaki, Japan, 7–10 October 2018; pp. 202–208.
27. Trecka, N.; van der Aalst, W.; Sidorova, N. Workflow completion patterns. In Proceedings of the 2009 IEEE International Conference on Automation Science and Engineering, Bangalore, India, 22–25 August 2009; pp. 7–12.
28. Zou, J.; Liu, X.; Sun, H.; Zeng, J. Live instance migration with data consistency in composite service evolution. In Proceedings of the 2010 6th World Congress on Services, Miami, FL, USA, 5–10 July 2010; pp. 653–656.
29. Xiang, D.; Liu, G.; Yan, C.G.; Jiang, C. A Guard-driven Analysis Approach of Workflow Net With Data. *IEEE Trans. Serv. Comput.* **2018**. [CrossRef]
30. Wisniewski, R.; Karatkevich, A.; Adamski, M.; Costa, A.; Gomes, L. Prototyping of Concurrent Control Systems With Application of Petri Nets and Comparability Graphs. *IEEE Trans. Control Syst. Technol.* **2017**, *26*, 575–586. [CrossRef]
31. Wisniewski, R.; Wisniewska, M.; Jarnut, M. C-exact Hypergraphs in Concurrency and Sequentiality Analyses of Cyber-Physical Systems Specified by Safe Petri Nets. *IEEE Access* **2019**, *7*, 13510–13522. [CrossRef]
32. McMillan, K.L. Using unfoldings to avoid the state explosion problem in the verification of asynchronous circuits. In *Computer Aided Verification*; Springer: Berlin/Heidelberg, Germany; 1992; pp. 164–177.
33. Franco, A.; Baldan, P. *True Concurrency and Atomicity: A Model Checking Approach with Contextual Petri Nets*; LAP LAMBERT Academic Publishing: Saarbrucken, Germany; 2015.
34. Haar, S. Types of asynchronous diagnosability and the reveals-relation in occurrence nets. *IEEE Trans. Autom. Control* **2010**, *55*, 2310–2320. [CrossRef]
35. Hickmott, S.L.; Rintanen, J.; Thiébaux, S.; White, L.B. Planning via Petri Net Unfolding. *Int. Jt. Conf. Artif. Intell.* **2007**, *7*, 1904–1911.
36. de León, H.P.; Saarikivi, O.; Kähkönen, K.; Heljanko, K.; Esparza, J. Unfolding Based Minimal Test Suites for Testing Multithreaded Programs. In Proceedings of the 15th International Conference on Application of Concurrency to System Design, Brussels, Belgium, 21–26 June 2015; pp. 40–49.
37. Khomenko, V.; Koutny, M. LP deadlock checking using partial order dependencies. In *International Conference on Concurrency Theory*; Springer: Berlin/Heidelberg, Germany, 2000; pp. 410–425.
38. Liu, G.; Reisig, W.; Jiang, C. A Branching-process-based method to check soundness of workflow systems. *IEEE Access* **2016**, *4*, 4104–4118. [CrossRef]

39. Rodriguez, C.; Schwoon, S. Verification of Petri nets with read arcs. In *International Conference on Concurrency Theory*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 471–485.
40. Dingle, N.J.; Knottenbelt, W.J.; Suto, T. PIPE2: A tool for the performance evaluation of generalised stochastic Petri Nets. *ACM SIGMETRICS Perform. Eval. Rev.* **2009**, *36*, 34–39. [CrossRef]
41. Heiner, M.; Herajy, M.; Liu, F.; Rohr, C.; Schwarick, M. Snoopy—A unifying Petri net tool. In *International Conference on Application and Theory of Petri Nets and Concurrency*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 398–407.
42. Jensen, K.; Kristensen, L.M.; Wells, L. Coloured Petri Nets and CPN Tools for modelling and validation of concurrent systems. *Int. J. Softw. Tools Technol. Transf.* **2007**, *9*, 213–254. [CrossRef]
43. Aalst, W.M.P.V.D.; Hee, K.M.V.; Hofstede, A.H.M.T.; Sidorova, N.; Wynn, M.T. Soundness of workflow nets: Classification, decidability, and analysis. *Form. Asp. Comput.* **2011**, *23*, 333–363. [CrossRef]
44. Liu, C.; Zeng, Q.; Cheng, L.; Duan, H.; Zhou, M.; Cheng, J. Privacy-preserving behavioral correctness verification of cross-organizational workflow with task synchronization patterns. *IEEE Trans. Autom. Sci. Eng.* **2020**. [CrossRef]
45. Xiang, D.; Liu, G.; Yan, C.; Jiang, C. DICER: Data Inconsistency CheckER based on the unfolding technique of Petri net. In Proceedings of the 2017 IEEE 14th International Conference on Networking, Sensing and Control (ICNSC), Calabria, Italy, 16–18 May 2017; pp. 115–120.
46. Saarikivi, O.; Ponce-De-León, H.; Kähkönen, K.; Heljanko, K.; Esparza, J. Minimizing test suites with unfoldings of multithreaded programs. *ACM Trans. Embed. Comput. Syst. (TECS)* **2017**, *16*, 45. [CrossRef]
47. Xiang, D.; Liu, G.; Yan, C.; Jiang, C. Detecting data-flow errors based on Petri nets with data operations. *IEEE/CAA J. Autom. Sin.* **2017**, *5*, 251–260. [CrossRef]
48. Xiang, D.; Liu, G. Checking Data-Flow Errors Based on The Guard-Driven Reachability Graph of WFD-Net. *Comput. Inform.* **2020**, *39*, 193–212. [CrossRef]
49. De Masellis, R.; Di Francescomarino, C.; Ghidini, C.; Tessaris, S. Enhancing workflow-nets with data for trace completion. In *International Conference on Business Process Management*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 89–106.
50. Evron, Y.; Soffer, P.; Zamansky, A. Incorporating data inaccuracy considerations in process models. In *Enterprise, Business-Process and Information Systems Modeling*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 305–318.
51. Lu, F.; Tao, R.; Du, Y.; Zeng, Q.; Bao, Y. Deadlock detection-oriented unfolding of unbounded Petri nets. *Inf. Sci.* **2019**, *497*, 1–22. [CrossRef]
52. Esparza, J.; Römer, S.; Vogler, W. An improvement of McMillan's unfolding algorithm. *Form. Methods Syst. Des.* **2002**, *20*, 285–310. [CrossRef]
53. Hillah, L.-M.; Kordon, F.; Petrucci, L.; Treves, N. Pnml framework: an extendable reference implementation of the petri net markup language. In Proceedings of the International Conference on Applications and Theory of Petri Nets, Braga, Portugal, 21–25 June 2010; pp. 318–327.
54. Aziz, M.W.; Rashid, M. Domain specific modeling language for cyber physical systems. In Proceedings of the 2016 International Conference on Information Systems Engineering (ICISE), Los Angeles, CA, USA, 20–22 April 2016; pp. 29–33.
55. Qi, L.; Zhou, M.; Luan, W. A two-level traffic light control strategy for preventing incident-based urban traffic congestion. *IEEE Trans. Intell. Transp. Syst.* **2018**, *19*, 13–24. [CrossRef]
56. Graja, I.; Kallel, S.; Guermouche, N.; Kacem, A.H. BPMN4CPS: A BPMN extension for modeling cyber-physical systems. In Proceedings of the 2016 IEEE 25th International Conference on Enabling Technologies: Infrastructure for Collaborative Enterprises (WETICE), Paris, France, 13–15 June 2016; pp. 152–157.
57. Flanagan, C.; Godefroid, P. Dynamic partial-order reduction for model checking software. *ACM Sigplan Not.* **2005**, *40*, 110–121. [CrossRef]
58. Lodde, A.; Schlechter, A.; Bauler, P.; Feltz, F. Data Consistency in Transactional Business Processes. In *International Conference on Business Informatics Research*; Springer: Berlin/Heidelberg, Germany, 2011; pp. 83–95.
59. Blanc, N.; Kroening, D. Race analysis for SystemC using model checking. *ACM Trans. Des. Autom. Electron. Syst. (TODAES)* **2010**, *15*, 1–32. [CrossRef]
60. Razavi, N.; Ivančić, F.; Kahlon, V.; Gupta, A. Concurrent test generation using concolic multi-trace analysis. In *Asian Symposium on Programming Languages and Systems*; Springer: Berlin/Heidelberg, Germany, 2012; pp. 239–255.
61. Sinha, N.; Wang, C. Staged concurrent program analysis. In Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering, Santa Fe, NM, USA, 7–11 November 2010; pp. 47–56.
62. Sun, S.X.; Zhao, J.L.; Nunamaker, J.F.; Sheng, O.R.L. Formulating the data-flow perspective for business process management. *Inf. Syst. Res.* **2006**, *17*, 374–391. [CrossRef]
63. Xiang, D.; Tao, X.; Liu, Y. An Incremental and Backward-Conflict Guided Method for Unfolding Petri Nets. *Symmetry* **2021**, *13*, 392. [CrossRef]
64. Kim, K.H.; Yavuz-Kahveci, T.; Sanders, B.A. JRF-E: Using model checking to give advice on eliminating memory model-related bugs. *Autom. Softw. Eng.* **2012**, *19*, 491–530. [CrossRef]
65. Zhang, M.; Wu, Y.; Shan, L.U.; Qi, S.; Ren, J.; Zheng, W. A Lightweight System for Detecting and Tolerating Concurrency Bugs. *IEEE Trans. Softw. Eng.* **2016**, *42*, 899–917. [CrossRef]