Detecting Infeasible Traces in Process Models

Zhaoxia Wang[1,2,3,4,5], Lijie Wen[1,3,4], Xiaochen Zhu[1,2,3,4], Yingbo Liu[1,3,4] and Jianmin Wang[1,3,4]
[1] *School of Software, Tsinghua University, Beijing 100084, China*
[2] *Department of Computer Science & Technology, Tsinghua University, Beijing 100084, China*
[3] *Key Lab for Information System Security, Ministry of Education, Beijing 100084, China*
[4] *National Laboratory for Information Science and Technology, Beijing 100084, China*
[5] *Logistical Engineering University, Chongqing 400016, China*

Keywords: Infeasible Traces, Process Models, Workflow Analysis.

Abstract: Workflow testing is an important method of workflow analysis in design time. A challenging problem with trace-oriented test data generation in particular and trace-based workflow analysis in general is the existence of infeasible traces for which there is no input data for them to be executed. In this paper we build on the theory of workflow nets and introduce workflow nets where transitions have conditions associated with them. We then demonstrate that we can determine which execution traces, that are possible according to the control-flow dependencies, are actually possible taking the data perspective into account. This way we are able to more accurately determine in design time the infeasible traces caused by the correlation between transition conditions along this trace. Finally, we provide a solution to automatically detecting the shortest infeasible trace.

1 INTRODUCTION

Workflow testing is an important method of workflow analysis in design time (van der Aalst and ter Hofstede, 2000). However, test data generation is an extremely complicated and time-consuming task as it requires a careful analysis and understanding of the source code and knowledge of the underlying concepts of the testing criterion. For this reason, automation of test data generation may provide a significant cost and time reduction for workflow testing. In trace-oriented test data generation, a set of traces is selected which satisfies one or more coverage criteria and test data are generated to exercise these traces. A challenging problem with trace-oriented test data generation in particular and trace-based workflow analysis in general is the existence of infeasible traces for which there is no input data for them to be executed. In this case, considerable effort might be wasted in trying to generate data for infeasible traces.

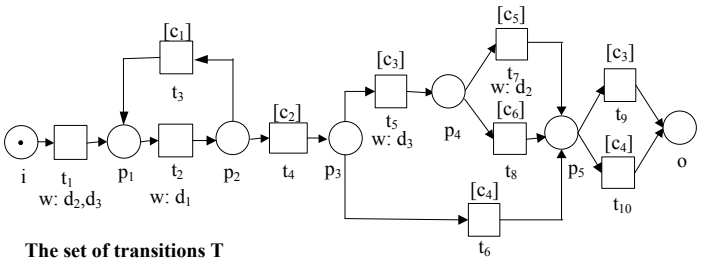
In software engineering domain, detection of branch condition and branch correlation is a kind of important methods in path feasibility analysis(Ngo and Tan, 2008). Here, inspired by the approaches of detecting the infeasible paths in software engineering domain we propose an approach to detect the infeasible traces in process model.

For instance, for the simplified example model for loan process (shown in Figure 1), the traces $t_1t_2(t_3t_2)*t_4t_5t_8t_{10}$ and $t_1t_2(t_3t_2)*t_4t_6t_9$ become infeasible when transition conditions are taken into account. The reason of traces $t_1t_2(t_3t_2)*t_4t_5t_8t_{10}$ non-existing is that the conjunctive transition conditions $c_3 \wedge c_4$ on transition t_{10} along $t_1t_2(t_3t_2)*t_4t_5t_8t_{10}$ are not satisfied, they are then non-executable (that is, they are **infeasible traces**).

In this paper, we build on the theory of workflow nets and introduce workflow nets where transitions have conditions associated with them. This way we are able to more accurately determine in design time the infeasible traces caused by the correlation between transition conditions.

The rest of this paper is organized as follows. First a formal definition of *workflow nets with transition conditions* or *WTC-nets* for short is given and we propose a reachability analysis approach based on the behavioural semantic of WTC-nets to detect the infeasible traces in process models (Section 2). We implement a tool to support the proposed approach and this tool is used to apply the approach to a number of real-life models (Section 3). Section 4 discusses the related work and Section 5 concludes the paper.

Wang Z., Wen L., Zhu X., Liu Y. and Wang J..
Detecting Infeasible Traces in Process Models.
DOI: 10.5220/0003989002120217
In *Proceedings of the 14th International Conference on Enterprise Information Systems* (ICEIS-2012), pages 212-217
ISBN: 978-989-8565-12-9



The set of transitions T

t_1: receive_application t_2: verify_application t_3: modify_application
t_4: check_loan amount t_5: check risk t_7: modify-loan
t_9: approve by manager t_{10}: approve by officer t_6, t_8: silent task

The set of condition variables

d_1: info_complete d_2: loan amount d_3: risk

The set of conditions

c_1: $\neg d_1$ c_2: d_1 c_3: d_2>50,000 c_4: d_2<=50,000 c_5: d_3 c_6: $\neg d_3$

Figure 1: A workflow model for the simplified loan process.

2 PRELIMINARIES

In this section we briefly introduce workflow nets with transition conditions (for short, WTC-nets). WTC-nets are based on Petri nets and workflow nets. In order to make the paper self-contained, some well-known definitions are introduced first.

2.1 Background

Below, we define Petri nets first. For an overview of Petri nets and an extensive bibliography the reader is referred to (Murata, 1989).

Definition 1 (Petri net)**.** *A Petri net N is a tuple (P,T,F) where P is a finite set of places, T is a finite set of transitions such that $P \cap T = \varnothing$, $P \cup T \neq \varnothing$, and $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.*

For each node x, i.e. a place or a transition, we use $\bullet x$ to denote its pre-set - the set of nodes which are input of x - i.e. $\bullet x = \{y|(y,x) \in F\}$, and $x\bullet$ for its post-set - the set of nodes which are output of x - i.e. $x\bullet = \{y|(x,y) \in F\}$.

Definition 2 (Marking)**.** *Let $N = (P,T,F)$ be a Petri net, a* marking *M of N is a function from its places to the set of natural numbers, i.e. $M : P \to \mathbb{N}$.*

Transitions can change the marking of a Petri net if they are fired. van der Aalst defines workflow nets by imposing certain syntactical restrictions on Petri nets (van der Aalst, 1998).

Definition 3 (Workflow net)**.** *A Workflow net $\mathcal{P}$ is a Petri net (P,T,F) such that there is a unique source place i, $\bullet i = \varnothing$, a unique sink place o, $o\bullet = \varnothing$, and such that every node $x \in P \cup T$ is on a path from i to o.*

The initial marking for a workflow net, or *WF-net* for short, corresponds to one token in its initial place and thus formally corresponds to i. The desired final marking is the marking that has one token in the output place o and no tokens in any other places.

2.2 Workflow Nets with Transition Conditions

Central to *WTC-nets* is the notion of a *condition*. Here we only informally describe condition class *cond*. A condition c is an instance of *cond*.

The expression form of the class *cond* is a logical expression. The **operators** used in *cond* include comparison operators (eg. le, leq, ge, geq, eq), arithmetical operators (eg. $plus, minus, times, frac, mod, div$), boolean operators (eg. $neg, and, or, implies$). The data **type** in *cond* contains *int*, *real* and *bool*. For a **variable** v used in a *cond*, the **domain** of v is a set of values that v can hold. The notion *var* is used as the set of all variables as well as the notion *VAR* is used as the set of all possible assignments of types to varianbles. In addition, function $V_{var}()$ is applied to a condition to yield the variables that are used in this condition.

A *WF-net* with transition conditions, or *WTC-net* for short, is a *WF-net* where the transitions are assigned conditions. We give an assumption that the variables that a transition can write to, may take on any value as a result of its execution.

Definition 4 (Workflow net with transition conditions)**.** *A workflow net with transition conditions $\mathcal{W}$ is a tuple $(\mathcal{P},V,C,G,W,TV)$, such that:*

- *$\mathcal{P} = (P,T,F)$ is a WF-net;*
- *$V \subseteq var$ is a set of net variables;*

- *$C \subseteq cond$ is a set of transition conditions with $\cup_{c \in C} V_{var}(c) \subseteq V$;*
- *$G : T \to C$ is a function assigning conditions to transitions;*
- *$W : T \to 2^V$ is a function that yields the variables that transitions can write to;*
- *$TV \in VAR$ is a function assigning a type to each net variable.*

Definition 5 (Marking of WTC-net)**.** *For a* WTC-net $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$*, a marking $\mathcal{M}$ is a tuple (M, σ), where M is a marking of* WF-net $\mathcal{P}$*, and $\sigma \in T^*$ is a sequence of transitions.*

The initial marking of this net $\mathcal{M}_0$ is defined as (i, ε), where i is the unique source place of $\mathcal{W}$, ε is empty sequence.

Subsequently we formally define whether transition condition is satisfied in the context of a particular assignment of values to variables. Given an assignment as an environment *e* and the set of all possible environments as *ENV*. The set of all possible values is denoted as *Val* and it is defined as the union of all integers, reals, and booleans, i.e. $Val = \mathbb{Z} \cup \mathbb{R} \cup \mathbb{B}$.

Definition 6 (Satisfiability of conditions)**.** *Let TV : $var \to Types$ be an assignment of types to variables and let c be a condition in the context of type assignment TV, i.e., $c \in cond$, $e : var \to Val$ an assignment of values to variables, and the function $V_{holds} : cond \times ENV \to bool$ can be applied to determine whether in the context of a given environment a condition holds.*

In order to evaluate if in the context of a transition sequence σ a given transition condition ϕ hold, we need to take into account the variables that are used in the transition conditions of transitions in σ and in ϕ as well as the order in which variables may be written by transitions in σ. To solve this problem, we introduce *fresh variables* as replacements for variables used in transition conditions of transitions whenever it turns out that these variables could have received a new value through the execution of preceding transitions. Fresh variables in a transition sequence σ take the form v_t^n where t is a transition which could write v, i.e. $v \in W(t)$, and n is a natural number, which for a particular occurrence of transition t in σ represents the number of previous occurrences of t in σ (this is required in order to be able to deal with loops). We assume that variables of the form v_t^n do not occur as net variables in a WTC-net, so that indeed they will be fresh whenever introduced. We also assume that given a renaming function $\psi : var \rightarrowtail var$ and a condition $c \in cond$, $c[\psi]$ denotes condition c where for every variable $v \in V_{var}(c) \cap \mathsf{dom}(\psi)$ each of its occurrences has been replaced by $\psi(v)$.

Definition 7 (Conjunctive transition conditions)**.** *Let $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ be a* WTC-net*, $\sigma \in T^*$ a transition sequence (possibly empty, i.e. $\sigma = \varepsilon$), $\psi : var \rightarrowtail var$ a renaming of variables, $G(t)[\psi]$ changing transition condition according to this renaming, $n : T \to \mathbb{N}$ an assignment of natural numbers to transitions, and b is any boolean expression. The conjunctive transition conditions along sequence with variables renaming $\rho^{\psi,\, n}(\sigma,\, b)$ is defined by:*

$$\rho^{\psi,\, n}(\varepsilon,\, b) = b$$

$$\rho^{\psi,\, n}(t\sigma,\, b) = \rho^{\psi',\, n'}(\sigma,\, b \wedge G(t)[\psi]) \text{ where}$$

$$\psi' = \psi \oplus \{(v,\, v_t^{n(t)}) \mid v \in W(t)\} \text{ and } n' = n \oplus \{(t,\, n(t)+1)\}$$

We are now in a position to formally define when a transition in a *WTC-net* is enabled in the context of a given marking.

Definition 8 (Enabled transition in WTC-net)**.** *Let $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ be a* WTC-net*, $\mathcal{M} = (M,\, \sigma)$ be a marking of this net, and $t \in T$ a transition in this net. Let b be a conjunctive transition conditions with variables renaming defined by $b = \rho^{\varnothing,\, \{(t,\, 0) \mid t \in T\}}(\sigma, true)$. Transition t is enabled in marking $\mathcal{M}$, $\mathcal{M}\,[t>$, iff an environment $e : V_{var}(b) \to Val$ can be found such that $V_{holds}(b,\, e,\, TV)$.*

Definition 9 (Firing)**.** *Let $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ be a* WTC-net*, $\mathcal{M} = (M, \sigma)$ be a marking of this net, and $t \in T$ an enabled transition. Firing transition t in marking $\mathcal{M}$ results in marking $\mathcal{M}' = (t\bullet \cup (M \setminus \bullet t), \sigma t)$. As per usual, we write $\mathcal{M} \xrightarrow{t} \mathcal{M}'$.*

Table 1: Deriving the conjunctive form of transition conditions along sequence $t_1t_2t_3t_2t_4$ for WTC-net in Figure 1.

TS[1]	VR[2]	TCT[3]	CTC[4]
t_1	$d_{2_{t_1}}^1, d_{3_{t_1}}^1$		
t_2	$d_{1_{t_2}}^1$		
t_3		$\neg d_{1_{t_2}}^1$	$\neg d_{1_{t_2}}^1$
t_2	$d_{1_{t_2}}^2$		$\neg d_{1_{t_2}}^1$
t_4		$d_{1_{t_2}}^2$	$\neg d_{1_{t_2}}^1 \wedge d_{1_{t_2}}^2$

[1] Transition Sequence.
[2] Variable Renaming.
[3] Transition Condition Transformation.
[4] Conjunctive Transition Conditions.

Example. Assuming for the WTC-net sample in the Figure 1, given a current marking $(p_2, t_1t_2t_3t_2)$, let us consider transition t_4 can be enabled or not. Firstly, we observe how to derive the conjunctive form of transition conditions for transition t_4 along the sequence $t_1t_2t_3t_2t_4$. As shown in the first column of Table 1, variable d_1 is written twice by transition t_2. Once each writing operation happens, d_1

is mapped into a new fresh variable as shown in the second column of Table 1. For example, variable d_1 is mapped into $d_{1_{t_2}}^1$ and $d_{1_{t_2}}^2$ respectively along the sequence $t_1t_2t_3t_2$. The corresponding transition condition is then changed based on the closest preceding variable renaming as shown in the third column in Table 1. For instance, the transition condition of transition t_3 in the sequence is changed as $\neg d_{1_{t_2}}^1$ following the closest preceding variable renaming $d_{1_{t_2}}^1$. The fourth column in Table 1 describes the dynamic evolution of the conjunctive form of transition conditions along the sequence. For the conjunctive form of transition conditions $\neg d_{1_{t_2}}^1 \wedge d_{1_{t_2}}^2$, there is at least an assignment of value *false* to $d_{1_{t_2}}^1$ and value *true* to $d_{1_{t_2}}^2$ which make the conjunctive form hold. Therefore, transition t_4 is enabled and then fired in the current marking $(p_2, t_1t_2t_3t_2)$.

Definition 10 (Reachable marking)**.** *Let* $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ *be a* WTC-net*, a marking* (M, σ) *with* $\sigma = t_1 \ldots t_n$ *is reachable iff a sequence of reachable markings* $M_0M_1 \ldots M_n$ *exists such that* $(i, \varepsilon) \xrightarrow{t_1} (M_1, t_1) \xrightarrow{t_2} (M_2, t_1t_2) \xrightarrow{t_3} \ldots \xrightarrow{t_{n-1}} (M_{n-1}, t_1 \ldots t_{n-1}) \xrightarrow{t_n} (M_n, t_1 \ldots t_n)$ *with* $M_n = M$.

With the behavioural semantics of a WTC-net system, we propose an extended coverability graph Algorithm based on that Finkel has proposed in (Finkel, 1991). The constructed graph of Figure 1 is shown in Figure 2 [An aside: any node with gray cycle is a dead node, node 2 (p_1, t_1) is the start state of the first cyclic execution t_2t_3, node 4 and node 9 labeled with $(p_1, \{(t_3, c_1)\})$ are the start state of the second round and third round of cyclic execution t_2t_3 respectively, Thus, the repeating of the cyclic executions is terminated in the start state of the third round by merging the node 9 to node 4 with the directed dotted arc].

2.3 Infeasible Trace of WTC-net

In this section, we will describe formally the semantics of the infeasible traces of WTC-nets.

Definition 11 (Infeasible Trace)**.** *Let* $\mathcal{W} = (\mathcal{P}, V, C, G, W, TV)$ *be a* WTC-net*,* (M, σ) *be a marking of* $\mathcal{W}$*,* t *be a transition with guard function* $G(t)$*,* $\sigma = t_1 \ldots t_n$ *is the* ***executable prefix*** *of transition* t *as well as trace* $\sigma \cdot t$ *is* ***infeasible****, iif:*

i. $M[t>$*, and*

ii. there is an environment $e : var \cup \mathsf{ran}(\psi) \rightarrowtail Val$ *such that for all* $1 \leq i \leq |\sigma|$*,* $V_{holds}(\rho^{\varnothing,\ \{(t,\ 0)\ |\ t \in T\}}(\sigma, true), e, TV)$*, and* $\neg(V_{holds}(\rho^{\varnothing,\ \{(t,\ 0)\ |\ t \in T\}}(\sigma.t, true), e, TV))$.

Definition 12 (Correlation between transition conditions)**.** *Given two transition conditions* c_1 *and* c_2 *assigned to guard function of* t_1 *and* t_2 *respectively. They are correlated, iif:*

(1) c_1 *and* c_2 *reference common condition variables, i.e, they have common condition variable set* $V_c = (v_1, v_2, \ldots, v_m)$*,*

(2) there is a trace from t_1 *to* t_2 *such that at least one variable in* V_c *isn't written.*

Subsequently, we propose a backward search algorithm on the generated coverability graph to search the shortest infeasible trace [An aside: in view of the limitation of space, for full details about the algorithms we proposed the readers are referred to http://laudms.thss.tsinghua.edu.cn/trac/Test/wiki/chengguo/ICEIS2012.pdf].

Figure 3 shows the result of this detecting algorithm. The set of infeasible traces $t_1t_2(t_3t_2)^*t_4t_5t_8t_{10}$ is caused by correlation relationship between transition condition c_3 on t_5 and transition condition c_4 on t_{10} as well $t_1t_2(t_3t_2)^*t_4t_6t_9$ caused by correlation relationship between transition condition c_4 on t_6 and transition condition c_3 on t_9.

3 EXPERIMENTS

We implemented the proposed approach for detecting the infeasible traces of WTC-nets. We made use of a well-known, open-source process analysis framework (ProM) (van der Aalst et al., 2009) and developed our tool as a ProM 6 analysis plug-in[1]. Figure 4 shows the returned result of detecting the infeasible traces in Figure 1.

Subsequently, we illustrate the applicability of our approach to real process models by carrying out experiments that are based on process models and data information related with transition conditions found in the TiPLM system[2]. These 29 collected real process models are divided into 4 types: engineering design and review (DR), process engineering and changing (PC), release management (RM), and application management (AM). Firstly, we transformed the TiPLM workflow models and the extracted data information from TiPLM database into WTC-net models. We then used the implemented tool to detect the infeasible traces.

[1]The ProM framework can be downloaded from http://www.processmining.org/

[2]TiPLM is a product life-cycle management solution, which is developed by THsoft InfoTech company (http://www.thit.com.cn) and widely used in Mainland China (well over 100 companies in the manufacturing industry in China adopted the TiPLM system).

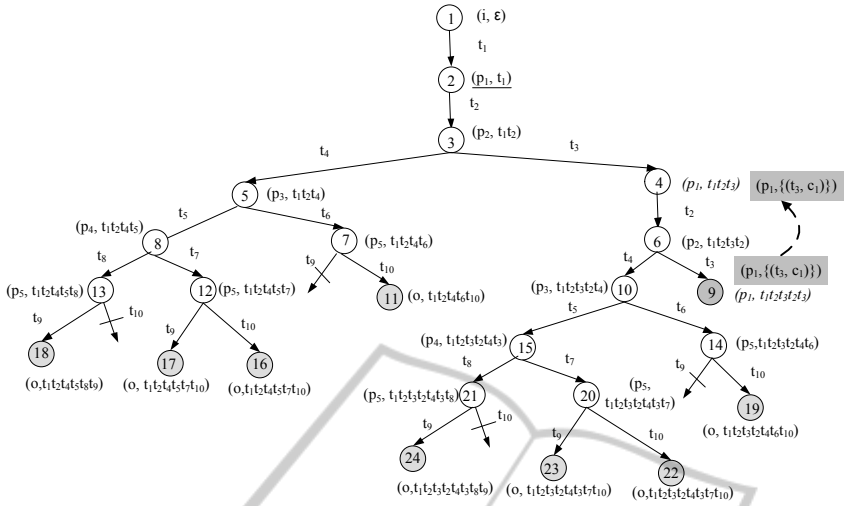
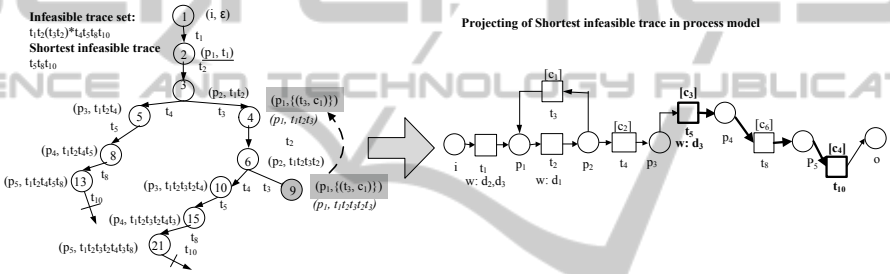
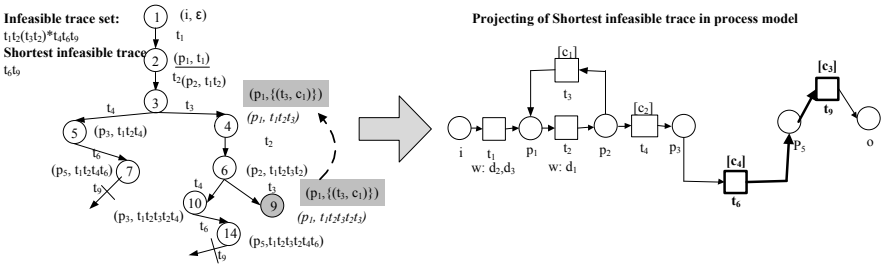


Figure 2: The coverability graph of the WTC-net in Figure 1.



(a) Sample 1 for shortest infeasible traces



(b) Sample 2 for shortest infeasible traces

Figure 3: Backwards detecting shortest infeasible trace.

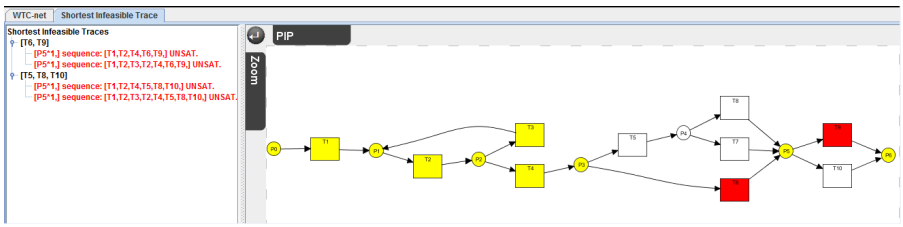


Figure 4: The screenshot of the detecting result of the shortest infeasible traces.

Table 2 shows that there are 8 (27.6%) process models among the 29 transformed WTC-net models. We compared the detection results with other measures including manual analysis. It turned out that our approach was always able to correctly detect the correlation relationship of the process definitions. All

Table 2: The detecting result of infeasible traces for real process models*.

PM	DR-1	DR-2	DR-3	DR-4	DR-5	DR-6	DR-7	DR-8	DR-9	DR-10
$SIT_{\#}$	0	2	26	0	0	0	1	0	0	0
PM	DR-11	PC-1	RM-1	RM-2	RM-3	RM-4	RM-5	RM-6	RM-7	RM-8
$SIT_{\#}$	11	0	0	2	1	0	0	0	0	0
PM	RM-9	RM-10	RM-11	RM-12	RM-13	AM-1	AM-2	AM-3	AM-4	
$SIT_{\#}$	0	0	0	1	0	0	1	0	0	

*PM describes process model, $SIT_{\#}$ denotes the number of the set of infeasible traces possessing the same shortest infeasible traces

examples for experiments can be downloaded from http://laudms.thss.tsinghua.edu.cn/trac/Test/wiki/chengguo/Test%20Data.zip.

4 RELATED WORK

In recent years, there are well-developed formalisms for workflow modeling taking data perspective into account based on WF-nets proposed in (van der Aalst, 1998).

In (Fan et al., 2007), a model called Dual Workflow Nets (DWF-nets) was proposed, which can explicitly model both the control flow and the data flow as well as their interactions. WFD-nets was proposed in (Trčka et al., 2009) to extend WF-nets with data elements for soundness verification. However, both of DWF-nets and WFD-nets don't consider the effects of concrete guard functions as well as their propagation explicitly.

For us, WTC-net is proposed for studying workflow nets where routing may be determined by transition conditions. Further, the shortest infeasible trace is detected based on the correlation between transitions while the works in (Fan et al., 2007) and (Trčka et al., 2009) focusing on the correctness of the auto execution of workflow models.

5 CONCLUSIONS

In this paper, we have defined workflow net with transition conditions *WTC-net* and its behavior semantics with the influence of transition condition in detail. In addition, we have presented an approach to detect infeasible trace and a tool has been implemented based on the approach.

In the future work, we will consider how to verify process model considering with transition condition in quantity.

ACKNOWLEDGEMENTS

This paper is supported by the 863 High-Tech Development Program of China (No. 2009AA043401), the National Science Foundation of China (No. 61003099, 61073005) and the National Science and Technology Major Project, China (No. 2010ZX01042-002-002-01).

REFERENCES

Fan, S., Dou, W., and Chen, J. (2007). Dual workflow nets: Mixed control/data-flow representation for workflow modeling and verification. In Chang, K. C.-C., Wang, W., 0002, L. C., Ellis, C. A., Hsu, C.-H., Tsoi, A. C., and Wang, H., editors, *Advances in Web and Network Technologies, and Information Management, APWeb/WAIM 2007 International Workshops: DBMAN 2007, WebETrends 2007, PAIS 2007 and ASWAN 2007, Huang Shan, China, June 16-18, 2007, Proceedings*, volume 4537 of *Lecture Notes in Computer Science*, pages 433–444. Springer.

Finkel, A. (1991). The minimal coverability graph for petri nets. In Rozenberg, G., editor, *Applications and Theory of Petri Nets*, volume 674 of *Lecture Notes in Computer Science*, pages 210–243. Springer.

Murata, T. (1989). Petri nets: Properties, analysis and applications. *Proceedings of the IEEE*, 77(4):541–580.

Ngo, M. N. and Tan, H. B. K. (2008). Heuristics-based infeasible path detection for dynamic test data generation. *Inf. Softw. Technol.*, 50(7-8):641–655.

Trčka, N., van der Aalst, W., and Sidorova, N. (2009). Data-flow anti-patterns: Discovering data-flow errors in workflows. In van Eck, P., Gordijn, J., and Wieringa, R., editors, *Advanced Information Systems Engineering, 21st International Conference, CAiSE 2009, Amsterdam, The Netherlands, June 8-12, 2009. Proceedings*, volume 5565 of *Lecture Notes in Computer Science*, pages 425–439. Springer-Verlag.

van der Aalst, W. (1998). The application of Petri nets to workflow management. *Journal of Circuits, Systems, and Computers*, 8(1):21–66.

van der Aalst, W. and ter Hofstede, A. H. M. (2000). Verification of workflow task structures: A Petri-net-baset approach. *Inf. Syst.*, 25(1):43–69.

van der Aalst, W., van Dongen, B., Günther, C., Rozinat, A., Verbeek, H., and Weijters, A. (2009). ProM: The process mining toolkit. In de Medeiros, A. and Weber, B., editors, *Proceedings of the Business Process Management Demonstration Track (BPMDemos 2009), Ulm, Germany, September 8, 2009*, volume 489 of *CEUR Workshop Proceedings*. CEUR-WS.org.