Formalizing a Decision Table into Petri Nets

Thanatta Mankong
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
6071007821@Student.chula.ac.th

Wiwat Vatanawood
Department of Computer Engineering
Chulalongkorn University
Bangkok, Thailand
wiwat@chula.ac.th

ABSTRACT

Business workflow designers use decision tables to lay out all possible business rules in the so-called IF-THEN format. However, as the business process continue to develop iteratively, the business rules are likely to explode considerably. This usually leads to an inadvertently error-prone design of the business process. Therefore, some designers employ a visualization technique to help them comprehend the decision tables more profoundly and mitigate the design errors. This paper proposes a scheme to visualize the decision tables using Petri nets. A Petri net is a formal method that is used to illustrate business flows and decision paths of the decision table. Our method starts with defining a set of rules which transforms a decision table into a Petri net model. Next, the table is transformed into a Petri net model. This transforming process often disclose inconsistent rules and redundant rules. After refining the table and obtaining a final model, a simulation tool is then used to animate the Petri net model. This allows the designers to verify the internal working of the decision table, identify its flaws, and fix the errors more precisely.

CCS Concepts

• Software and its engineering→Software organization and properties→Software system structures→Software system models→Petri nets • Software and its engineering→Software organization and properties→Software functional properties →Formal methods →Model checking.

Keywords

Decision Table; Petri Nets; Transformation Rule; Formal Model

1. INTRODUCTION

The business process and business rule design of an enterprise application are an essential process. It becomes more complex as the application grows. The business rules are designed to be easily changed by not hardwired into the source code. In practice, a decision table is one of the alternatives and most popular methods that is used to mange a huge set of business rules by providing the tabular form of the IF-THEN rules or cause and effect rules. A decision table is a good way to cope with the various combination of conditions which activate the different actions. In addition, the decision table is used in software testing technique which is used to generate relevant test cases [1].

ICSEB 2019, December 9–11, 2019, TOKYO, Japan
© 2019 Association for Computing Machinery.
ACM ISBN 978-1-4503-7649-5/19/12…$15.00
https://doi.org/10.1145/3374549.3374571

Several enterprise applications tend to consist of multiple complex business flows along with the corresponding business rules. In short, there is be a large combination of business conditions and consequences of large business actions in the decision tables, which is the cause of redundancy and inconsistency in the business rules.

Therefore, it is beneficial if the decision table can be simulated to give the business process designer an insight into its behavior before developing the system, using the formal methods. A formal representation of the decision table, called Petri net, is selected. It simulates the behavior of the cause and effect using a Petri net simulation tool. The Petri net reachability graph shows all possible state space of its behavior and is used to analyze the desirable properties.

Prior studies proposed method to utilize the original Petri nets and its extended version of Petri nets, such as colored Petri nets, as the formal models of the decision tables and decision rules. For example, [2] proposed the description of the decision table in Petri net models. [3] adopt the colored Petri nets to support decision making features of the decision making system, while [4] implemented the business rules defined by the given decision table in ML functions and automatically inserted these functions as the arc inscriptions in an original colored Petri nets. Also, [5] showed the knowledge representation of the complex expert systems in terms of rules and represented these rules using the Petri nets as the dynamic system representation and rule derivation. Whilst, [6] proposed the representation of the system requirements in high level Petri nets and intended to generate the test cases of these requirements from the resulting Petri nets.

Since the previous studies do not check the validity of the rules specified in the decision table, we propose a scheme to formalize a given decision table into a corresponding Petri net. The inconsistency and redundancy of the business rules are alerted by the simulation of the resulting Petri net. This paper is organized as follows. Section 1 is the introduction and section 2 describes briefly about backgrounds of the decision table and Petri nets. Section 3 describes the formalization scheme. Section 4 shows a case study and our conclusion is discussed in section 5.

2. BACKGROUND

2.1 Decision Table

A decision table [7, 8] is a tabular form which presents conditions and actions of the system and also expresses the business rules or business logics in IF-THEN format. Since a decision table collects all possible essential decision rules needed in a particular business transaction in the system, it is typically used in the software testing in order to demonstrate the input and output data from the collaboration of conditions and the consequence of the actions. The decision table is a general technique that is suitable for the complex system or various rules of decision making, which can clearly distribute conditions and decisions [9]. A decision table consists of four major components: condition stubs, condition entries, action stubs and action entries. The fundamental parts of a simple decision table are shown in Table 1 and the expanded version of the “Don’t care” conditions are shown in Table 2.

In Table 1, the condition entries and action entries allow the following symbols:

- Symbol “T” implies that the condition is true.
- Symbol “F” implies that the condition is false.
- Symbol “–” implies that the condition is “Don’t care” which is either true or false.
- Symbol “X” implies that the action is performed.

Table 1. Portions of a decision table [9]

Condition Stubs	**Condition Entries**					
	Rule 1	**Rule 2**	**Rule 3**	**Rule 4**	**Rule 5**	**Rule 6**
C1	T	T	T	F	F	F
C2	T	T	F	T	T	F
C3	T	F	–	T	F	–
Action Stubs	**Action Entries**					
A1	X	X		X		
A2	X				X	
A3		X		X		
A4			X			X

There are two problems that are commonly encountered in generating a decision table. Firstly, the inconsistency of the rules may happen if there are two rules sharing the same conditions, but the actions are conflict. If the system has two decisions, it may lead to wrong conclusions [10, 11]. Secondly, the redundancy of the rules may happen if there are two rules sharing the same conditions and the actions. The redundant rule seems to be harmless but problems may occur during maintenance [11].

In Table 1, the symbol “–” so called “Don’t care” conditions shown in rules 3 and rule 6 are expanded to rule 3.1, 3.2 and rule 6.1, 6.2 in Table 2. As mentioned earlier, the “Don’t care” condition means the condition can be either true or false so that it causes the expanding of the existing rule. More rules are concerned.

Table 2. Expanded Version of the “Don’t care” condition [9]

Condition Stubs	**Condition Entries**							
	Rule 1	**Rule 2**	**Rule 3.1**	**Rule 3.2**	**Rule 4**	**Rule 5**	**Rule 6.1**	**Rule 6.2**
C1	T	T	T	T	F	F	F	F
C2	T	T	F	F	T	T	F	F
C3	T	F	**T**	**F**	T	F	**T**	**F**
Action Stubs	**Action Entries**							
A1	X	X			X			
A2	X					X		
A3		X			X			
A4			X	X			X	X

2.2 Petri Nets

Petri net [12] is a directed bipartite graph that is used to describe the behavior of a system by showing the initial state and the consequent states of the system during operation. The reachability graph of a Petri net shows all possible states of a system. A Petri net contains two types of nodes called place and transition, and the arcs connecting between nodes as shown in Figure 1.

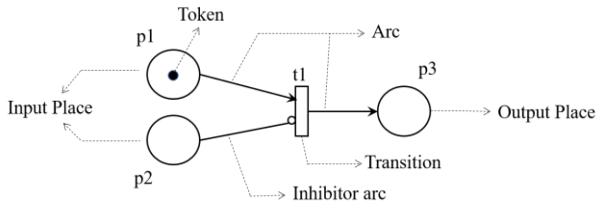


Figure 1. A sample of Petri net [12]

A Petri net is 6-tuple $PN = (P, T, F, W, M, M_0)$

- P is the finite set of places.
- T is the finite set of transition.
- $F \subseteq (P \times T) \cup (T \times P)$ is the flow relation.
- $W: F \rightarrow \mathbb{N}$ is the function weight of flow relation.
- $M: P \rightarrow \mathbb{N}$ is a marking of a Petri net (graph) as a multiset of its places.
- M_0 is the initial marking.

The place nodes of the Petri net represent the distributed states by holding tokens. Whilst, the transition nodes represent the consuming and producing of the tokens between input places and output places. A transition is enabled if sufficient tokens are available in each input place. When an enabled transition fires tokens, the tokens of each input place are consumed and those of each output place are produced. If there are more than one enabled transition, then only one of those is selected to fire. The flow relation of the Petri net is represented using arcs/inhibitor arcs to connect place and transition. The weight labelled on the arc represents number of consuming tokens and producing tokens. A token is a black dot that marks in the place, representing the working state. Tokens marked in the initial state is called the initial marking. In Petri net model, there are four main patterns which are sequential, concurrent, conflict (choice) and synchronization patterns as shown in Figure 2.

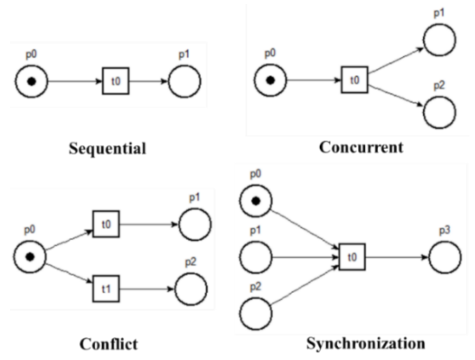


Figure 2. Samples of main patterns of Petri nets [12]

3. OUR FORMALIZING APPROACH

In this section, our formalizing approach is illustrated in four steps, as shown in Figure 3. However, the formal definitions of a decision table and the target Petri net must be defined beforehand.

Our first step begins with designing mapping rules to formalize the decision table using the syntax of the Petri net. Next, a given original decision table is imported and the mapping rules is applied to generate a Petri net. Finally, the Petri net is verified.

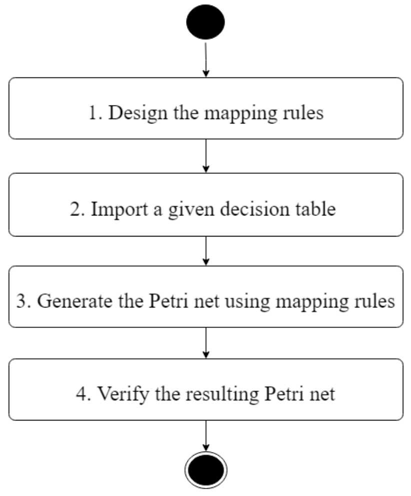

1. Design the mapping rules
2. Import a given decision table
3. Generate the Petri net using mapping rules
4. Verify the resulting Petri net

Figure 3. Our Formalizing Approach

Definition 1 Decision Table
A decision table is defined as 7-tupe *DT* = (*C, A, R, CTR, CFR, CDR, AR*) where:

- *C* is a set of conditions in the condition stub.
- *A* is a set of actions in the action stub.
- *R* is a set of rules which specifies the truth values of the related conditions and the related actions to be performed. The truth values include true (denoted by symbol "T"), false (denoted by symbol by "F"), and "Don't care" (denoted by "–"). The actions to be performed are specified by symbol "X".
- $CTR: R \rightarrow C_T$ is a function that specifies a set of conditions $C_T \in C$ which contains symbol "T"
- $CFR: R \rightarrow C_F$ is a function that specifies a set of conditions $C_F \in C$ which contains symbol "F"
- $CDR: R \rightarrow C_D$ is a function that specifies a set of conditions $C_D \in C$ which contains symbol "–"
- $AR: R \rightarrow A_P$ is a function that specifies a set of actions $A_P \in A$ which contains symbol "X"

For example, the decision table shown in Table 1 is defined as follows.

- *C = {C1, C2, C3}*
- *A = {A1, A2, A3}*
- *R = {R1, R2, R3, R4, R5, R6}*
- *CTR(R1) = {C1, C2, C3}, CTR(R2) = {C1, C2}, CTR(R3) = {C1}, CTR(R4) = {C2, C3}, CTR(R5) = {C2}, CTR(R6) = Ø*
- *CFR(R1) = Ø, CFR(R2) = {C3}, CFR(R3) = {C2}, CTFR(R4) = {C1}, CFR(R5) = {C1, C3}, CFR(R6) = {C1,C2}*
- *CDR(R1) = Ø, CDR(R2) = Ø, CDR(R3) = {C2}, CDFR(R4) = Ø, CDR(R5) = Ø, CDR(R6) = {C3}*
- *AR(R1) = {A1, A2}, AR(R2) = {A1, A3}, AR(R3) = {A4}, AFR(R4) = {A1, A3}, AR(R5) = {A2}, AR(R6) = {A4}*

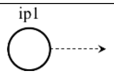
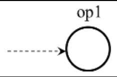
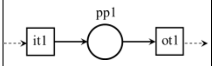
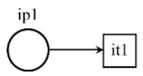
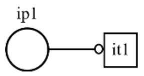
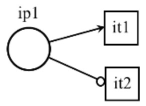
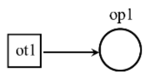
Definition 2 Pattern of Petri net for a decision table
Given a Petri net *PN* [12] and a decision table *DT*, a pattern of the Petri net for the given decision table is defined as a 9-tuple *DTNet* = (*IP, IT, PP, OT, OP, IF, PF, OF,* M_0) where:

- *IP* is a finite set of outer input places which represent the condition stub *C* in decision table *DT*. Any token in an outer input place means the input truth value of true (denoted by symbol "T") of a condition $c \in C$.
- *IT* is a finite set of transitions for combining the outer input places in *IP*.
- *PP* is a finite set of process places which represent the combinations of the truth values found in the rules. Any token in a process place $pp \in PP$ means the truth values of the condition stub matching a rule $r \in R$ in decision table *DT*.
- *OT* is a finite set of output transitions which fires a token to the outer output places.
- *OP* is a finite set of outer output places which represents the final actions to be performed.
- $IF: (IP \times IT)$ is the input relation that defines directed arcs/inhibitor arcs from outer input places to their transitions.
- $PF \subseteq (IT \times PP) \cup (PP \times OT)$ is the flow relations of fan-in and fan-out flows of the process places in *PP*.
- $OF: (OT \times OP)$ is the output relation that defines directed arcs between output transitions and outer output places.
- M_0 is the initial marking.

3.1 Design the Mapping Rules

In this step, we define a set of mapping rules to formalize a given decision table into a completely relevant Petri net. Each element of the decision table *DT* is considered to generate a set of corresponding elements of a Petri net *PN*. The mapping rules are shown in Table 3.

Table 3. Our mapping rules to formalize a decision table into Petri net

Decision table	Pattern Element Name	Petri net Notation
Condition Stub	Input place (*IP*)	
Action Stub	Output place (*OP*)	
Rule	Process place (*PP*) with Input transition (*IT*) and Output transition (*OT*)	
True condition "T"	Arc of the true condition in the input relation (*IF*)	
False condition "F"	Inhibitor arc of the false condition in the input relation (*IF*)	
Don't care condition "–"	Both arc and inhibitor arc of the "Don't care" condition in the input relation (*IF*)	
Action Entries "X"	Arc of the performed actions in the output relation (*OF*)	

As shown in Table 3, our mapping rules are described as follows.

1) Mapping input place

For each condition in the given condition stub $c \in C$, an input place $ip \in IP$ is generated. The number of input places in *IP* is equal to the number of the given conditions in condition stub *C*.

2) Mapping output place

For each performed actions $a \in A_P$, an output place $op \in OP$ is generated. Practically, the unperformed actions are not relevant to the Petri net.

3) Mapping process place

For each rule $r \in R$, a process place $pp \in PP$ is generated along with both connecting input transition $it \in IT$ and output transition $ot \in OT$. The input transition *it* fires and produces a token to the process place *pp* of the rule *r* whenever the rule *r* is satisfied with the defined conditions.

4) Mapping an arc of the true condition

For each condition in the given condition stub $c \in C$ and $c \in C_T$ where C_T is s set of the conditions containing the symbol "T", an arc $(ip, it) \in IF$ is generated to connect from an input place *ip* to the associate input transition *it*. The input transition *it* is the one with the process place *pp* of the rule *r* where $CTR(r) = C_T$.

5) Mapping an inhibitor arc of the false condition

For each condition in the given condition stub $c \in C$ and $c \in C_F$ where C_F is the set of the conditions containing the symbol "F", an inhibitor arc $(ip, it) \in IF$ is generated to connect from an input place *ip* to the associate input transition *it*. The input transition *it* is the one with the process place *pp* of the rule *r* where $CFR(r) = C_F$. The inhibitor arc prohibits the firing of the transition if there is an existing token in the input place.

6) Mapping the "Don't care" condition

For each condition in the given condition stub $c \in C$ and $c \in C_D$ where C_D is the set of the conditions containing the symbol "–", the process place *pp* of the rule *r* where $CDR(r) = C_D$ along with its associate input transition *it* and output transition *ot* are duplicated as input transitions *it1, it2* and output transitions *ot1, ot2* respectively. After that, an arc $(ip, it1) \in IF$ is generated to connect from an input place *ip* to the associate input transition *it1*. Then, an inhibitor arc $(ip, it2) \in IF$ is generated to connect from an input place *ip* to the associate input transition *it2*. In fact, the rule *r* with "Don't care" condition is explicitly expand into two rules with both true and false values for the "Don't care" symbol. However, the number of the splitting rules depend on the number of the "Don't care" conditions found in the same rule.

7) Mapping the arc of the performed action

For each performed action $a \in A_P$, an arc $(ot, op) \in OF$ is generated. The output transition *ot* is the one with the process place *pp* of the rule *r* where $AR(r) = A_P$.

3.2 Import a Given Decision Table

A business decision in workflow is represented as a decision table in tabular format. In our approach, a spreadsheet of the decision table is accepted and imported. The proper layout of the decision table is shown in Table 4.

Table 4. The layout of a given decision table in tabular form

Condition Stub_ID	**Condition Stub_Description**	**Rule_ID**			
		R1	**R2**	**R3**	**R4**
C1	Condition1	T	T	T	F
C2	Condition2	T	T	F	–
C3	Condition3	T	F	F	F
Action Stub_ID	**Action Stub_Description**	**Action Entries**			
A1	Action1	X		X	
A2	Action2		X		X

As shown in table 4, a given decision table is formally considered as a decision table *DT* where the condition stub *C = {C1, C2, C3}* and the action stub *A = {A1, A2}*. There are four rules in *R = {R1, R2, R3, R4}* where *CTR(R1) = {C1, C2, C3}, CTR(R2) = {C1, C2}, CTR(R3) = {C1}, CTR(R4) = Ø, CFR(R1) = Ø, CFR(R2) = {C3}, CFR(R3) = {C2, C3}, CFR(R4) = {C1, C3}, CDR(R1) = Ø, CDR(R2) = Ø, CDR(R3) = Ø, CDR(R4) = {C2}*. The action entries in *AR* are *AR(R1) = {A1}, AR(R2) = {A2}, AR(R3) = {A1}, AR(R4) = {A2}.*

3.3 Generate the Petri Net Using Mapping Rules

In this step, we illustrate the formalizing of a decision table into a Petri net using mapping rules in Table 3. The resulting Petri net is called *DTNet* which is defined as a pattern of Petri net generated from a decision table. The sample of decision table given in Table 4 is formalized into the resulting Petri net *DTNet* shown in Figure 4.

Given a decision table *DT*, the step-by-step description of our formalization approach is as follows.

1) Generating the outer input places *IP* from the condition stub *C* of a decision table *DT*.

The resulting Petri net contains the outer input places which are generated for every condition $c \in C$. From the given set of condition stub *C = {C1, C2, C3}*. The set of input places *IP = {ip1, ip2, ip3}* is generated.

2) Generating the process places *PP* from the rule *R* of a decision table *DT* and associate.

A particular process place $pp \in PP$ is generated for a rule $r \in R$ of decision table *DT*. For each process place *pp*, there exist one associative input transition $it \in IT$ and an arc $(it, pp) \in PF$. As shown in Figure 4, {(*it1, pp1*), (*it2, pp2*), (*it3, pp3*), (*it4, pp4.1*), (*it5, pp4.2*)} $\subseteq$ *PF* is generated. After that, there are one associative output transition $ot \in OT$ and an arc $(pp, ot) \in PF$. In Figure 4, {(*pp1, ot1*), (*pp2, ot2*), (*pp3, ot3*), (*pp4.1, ot4*), (*pp4.2, ot5*)} $\subseteq$ *PF* is generated, as well.

3) Generating the input relation $IF: IP \times IT$ for *DTNet.*

For each $c \in CTR(r)$ where $r \in R$, one arc is generated to connect from the input place c to transition $ip \in IP$ where *ip* is the input transition of the process place for rule $r \in R$. Whilst, for each $c \in CFR(r)$ where $r \in R$, one inhibitor arc is generated to connect from the input place c to inhibit the transition $ip \in IP$ where *ip* is the input transition of the process place for rule $r \in R$. In case of "Don't care" conditions, for each $c \in CDR$ where $r \in R$, the input transition *it* and the process place for rule *r* are duplicated as transitions *it1, it2* and process places *pp1, pp2* in order to handle both true and false value of the "Don't care" condition $c \in CDR$.

One input transition *it1* deals with true value while the other transition *it2* deals with false value.

In Figure 4, the arc (*ip2, it4*) and the inhibitor arc (*ip2, it5*) are generated for the "Don't care" condition $C3 \in CDR(R4)$.

4) Generating the outer output places *OP* for *DTNet*.

The outer output places *OP* are generated for every action $a \in AR(r)$ where $r \in R$ in the given decision table *DT*. In the Figure 4, $\{op1, op2\} \in OP$ is generated. For each $a \in AR(r)$ where $r \in R$, an arc $of \in OF$ is generated to connect from $ot \in OT$ of the rule *r* to the outer output place $op \in OP$ of the action $a \in AR(r)$.

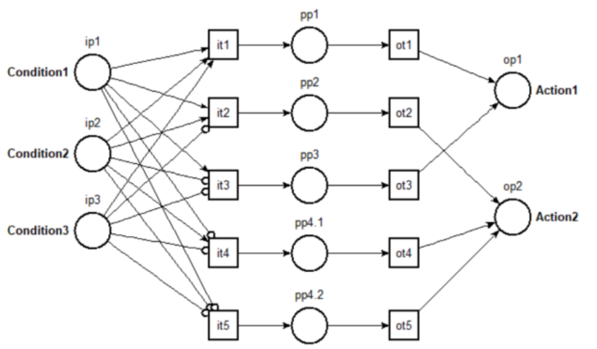


Figure 4. A resulting *DTNet* of our sample decision table

3.4 Verify the Resulting Petri Net

The resulting Petri net *DTNet* is verified and simulated by using TINA tool (Petri net simulation tool) [13]. The system behavior can be observed in the step-by-step manner or the random animated manner. The observed behavior reflects all decision paths of the decision table. The Petri net is verified against the decision table to look for inconsistent results between them. Furthermore, the linear temporal logic formulas of the desirable properties, including the reachability, deadlock, or livelock of the Petri net, are used to verify the resulting *DTNet*.

4. CASE STUDY

In this section, our formalization scheme is demonstrated by using the mapping rules. This case study is a part of the debt collection strategy on automotive leasing system. A sample decision table of the debt collection strategy is presented in Table 5.

Table 5. A sample of decision table of the debt collection strategy

Condition Stub_ID	**Condition Stub_Description**	**Rules**					
		R1	**R2**	**R3**	**R4**	**R5**	**R6**
C1	Customer grade is A	T	T	T	T	–	T
C2	Customer grade is B	F	F	F	F	–	F
C3	On Due date	T	F	F	F	F	F
C4	3-7 days overdue	F	T	F	F	F	F
C5	8-30 days overdue	F	F	T	F	F	–
C6	31-60 days overdue	F	F	F	T	F	F
C7	61+ days overdue	F	F	F	F	T	F
Action Stub_ID	**Action Stub_Description**	**Action entries**					
A1	SMS	X					
A2	Voice by Inhouse Collector		X	X			
A3	Voice by Outsource Collector				X		X
A4	Letter					X	
A5	Field Collector Agent					X	

First, a decision table is defined as shown in Table 5. A Petri net graph is generated by TINA tool and transformation rules to simulate the table's dynamic behavior.

DTNet is generated for the collection strategy on automotive leasing system in Figure 5.

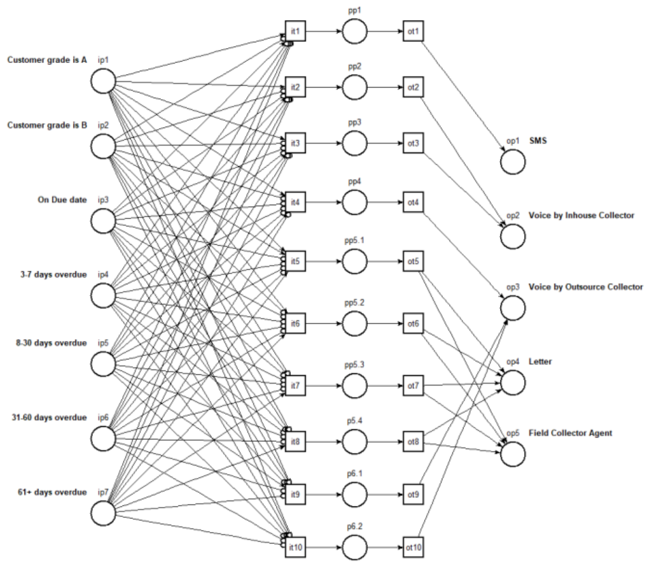


Figure 5. *DTNet* for the collection strategy on automotive leasing system

The given decision table in Table 5, the "Don't care" conditions appear in rule number 5 and rule number 6. These rules must be expanded to cope with the possibility of both symbol "T" and symbol "F" conditions. In practice, "Don't care" condition means that its truth value can be either true or false. In our case study, the rule number 5 can be expanded and the process places *pp5.1, pp5.2, pp5.3, pp5.4* along with their associate input transitions and output transitions are generated. As mentioned earlier, the arcs and inhibitor arcs for the "Don't care" conditions are also determined.

Secondly, the resulting Petri net *DTNet* , Figure 5, of the decision table as depicted in Table 5 is now ready for simulating in the Petri net tools. Our proposed *DTNet* is expected to have only one enabled input transition at a time, which means only one rule is satisfied at a time. Whenever, more than one enabled input transitions are found in the simulation, there must be inconsistent or redundant rule. The *DTNet* shows the marking of the enabled input transition during the simulation so that the decision table designer can visualize and identify the suspect rules before applying the table in the proceeding phases.

Next, the sample of the inconsistent rule found in this case study because of the input transitions *it3* and *it9*, which are highlighted in red as shown in Figure 6, indicate the inconsistent rules of the decision table. This also prompts us that the two transitions are enabled simultaneously. The input transition *it3* connecting to the process place *pp3* of rule number 3 and the input transition *it9* connecting to the process place *pp6.1* of rule number 6, are also considered inconsistent. Then, the given decision table must be corrected. And, its the condition truth values of rule number 3 and rule number 6 must be redefined.

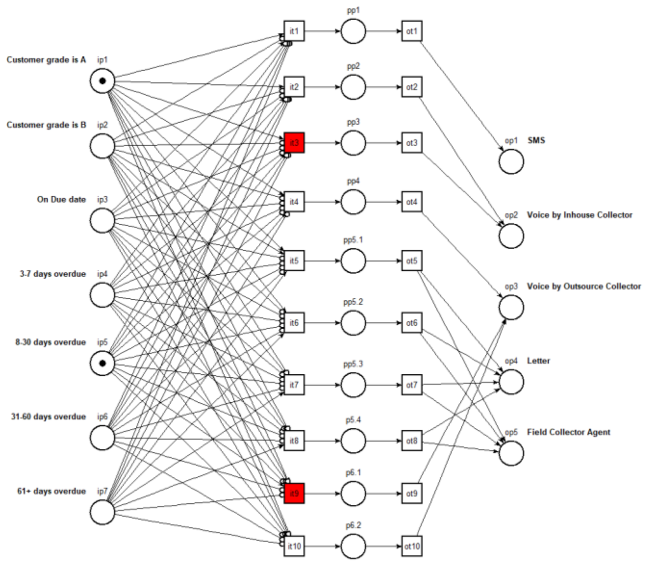


Figure 6. Sample of the inconsistency rule of decision table.

Finally, as we have simulated all decision path, both step-by-step and random methods, and found no further problems with the rules, the designer is informed to update the decision table.

5. CONCLUSION

In our research, we design a set of mapping rules to transform a decision table into a formal pattern. This pattern represents dynamic behavior of the decision table and is used to analyze its decision paths. The pattern is also used to check for inconsistent and redundant rules in the decision table. We propose an alternative of formalizing the decision table into a Petri net using a set of mapping rules. The resulting Petri net *DTNet* is defined to ease the automatically generated. A set of Petri net patterns for decision table is described as input places for representing the condition stub in decision table, output places for the action stub. Whilst, the rules of the decision table are defined using the process places along with its both input transitions and output transitions. The truth value of the conditions including “Don’t care” are defined as the connecting arcs and inhibitor arcs between the input places and input transitions. The performed actions are defined using the arcs between the output transitions and output places. The resulting Petri net is simulated to visually observe the inconsistency and redundancy of the rules. The LTL formulas of the desirable properties of the Petri net could be verified as well using the TINA Petri net tool.

6. REFERENCES

[1] Masuda, S., Matsuodani, T., and Tsuda, K. 2016. Syntactic Rules of Extracting Test Cases from Software Requirements. In *Proceedings of the 2016 8th International Conference on Information Management and Engineering* (Istanbul, Turkey, November 02 - 05, 2016). CHI '00. ACM, New York, NY, 12-17.

[2] Szpyrka, M., and Szmuc, T. 2007. Decision Tables in Petri Net Models, Springer, 2007, 648–657.

[3] Guibu, H. I., and Neto, J. J. 2017. Decision making system supported by Adaptive Coloured Petri Nets. In *Proceedings of the International Journal of Computer Networks & Communications (IJCNC),*(Australia, July, 2017).

[4] Deesukying, J., and Vatanawood, W. 2016. Generating of Business Rules for Coloured Petri Nets. In *Proceedings of the 2016 IEEE/ACIS 15th International Conference on Computer and Information Science (ICIS).*

[5] Tavana, M. 2008. Knowledge-Based Expert System Development and Validation with Petri Nets, Journal of Information & Knowledge Management (JIKM), 2008 , 37–46.

[6] Desel, J., Oberweis, A., and Zimmer, T. 1997. A Test Case Generator for the Validation of High-Level Petri Nets. In *Proceedings of the 1997 IEEE 6th International Conference on Emerging Technologies and Factory Automation Proceedings, EFTA '97.*

[7] Shamim, A., Hussain, H., and Shaikh, H. U. 2010. A Framework for Generation of Rules from Decision Tree and Decision Table. In *Proceedings of the 2010 International Conference on Information and Emerging Technologies* (Karachi, Pakistan, June 14-16, 2010).

[8] Auechaikul, T., and Vatanawood, W. 2007. A Development of Business Rules with Decision Tables for Business Processes. In *Proceedings of the International Conference of IEEE TENCON.*

[9] Jorgensen, C., P. 2014. *Software Testing A Craftsman's Approach Fourth Edition,*117-131.

[10] Ma, J., Lu, J., and Zhang, G. 2007. A Rule-Map based Technique for Information Inconsistency Verification, In *Proceedings of the* 2007 *Information, Decision and Control* (Adelaide, Qld., Australia, February 12-14, 2007).

[11] McGuire, J. G. 1990. Uncovering redundancy and rule-inconsistency in knowledge bases via deduction. In *Proceedings of the Fifth Annual Conference on Computer Assurance, Systems Integrity, Software Safety and Process Security.*

[12] Reisig, W. 2013. Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, Springer.

[13] TIme Petri Net Analyzer. [Online] Available: http://projects.laas.fr/tina/index.ph