International Journal of Engineering Trends and Technology
ISSN: 2231 – 5381 / https://doi.org/10.14445/22315381/IJETT-V71I2P248

Volume 71 Issue 2, 466-492, February 2023
© 2023 Seventh Sense Research Group®

Original Article

Improving Software Requirements - Analysis of Petri Net Models for Inconsistency and Incompleteness

A Keshav Bharadwaj[1], Vinod K Agrawal[2], Jayashree R[3]

[1,3]Department of Computer Science, PES University, Karnataka, India.
[2]R & D, Dayananda Sagar University,Karnataka, India.

[1]Corresponding Author : keshavbharadwaj@pes.edu

Received: 05 October 2022 Revised: 15 January 2023 Accepted: 31 January 2023 Published: 25 February 2023

Abstract - *Stakeholders of most software projects express requirements in natural language. (English is taken as the natural language here). To minimize errors in the requirements, an analysis of requirements is necessary. Direct analysis of requirements expressed in natural language is complex. Hence it is preferable that they be expressed using some formal method before they are analyzed and used for specifications development. In this paper, we have made the assumption that specification documents and Petri net models of the software requirements are available to us. We propose methods to analyze these Petri net models for inconsistency and incompleteness in the software requirements. Results obtained from the analysis are used to redefine the given user requirements such that inconsistency and incompleteness are removed. The example of an Automated Teller Machine has been used to demonstrate our approach.*

Keywords - *Analysis, Completeness, Consistency, Petri nets, Requirements.*

1. Introduction

Successful software project completion largely depends on good requirements and specifications. The development of consistent and complete requirement specifications is an iterative process. It involves analysis and incorporation of feedback from the analysis done. Natural language (NL), i.e., the English language, is generally used to express requirements as it is the most widely spoken.[1] Yet the requirements provided by users are often ambiguous and incomplete and rely on implicit information.[2] Such informally expressed requirements are not easy to analyse. It is, therefore, necessary to translate user requirements expressed in NL using one of the many formal techniques [3,4] so that they can be analysed for consistency and completeness, among other properties. Though these formal methods are difficult and costly to use, among other disadvantages [3], they help create precise, unambiguous specifications and thereby enable reasoning and analysis.[4]

The use of formal methods like State charts, Alloy, etc., for developing requirement specifications, has been practiced by several researchers.[5,6,7,8] But these methods are mostly manual and quite complex.[5,6,7,8] In the case of Petri nets (PNs), there is not only scope for automation and tool development, but they are also relatively easier to analyse.[9,10]In [11], the many advantages of using PNs have been brought out well. Using PNs, some approaches have been put forth to check the inconsistency of requirements, but most are difficult to apply.

Secondly, very little research has been done using PNsto to analyse incompleteness[10,12,13,14,15] in requirements. Inconsistency and incompleteness in requirements are a sure cause for schedule and cost overruns in a project. Thus, there is a need for a simple, easy methodology that analyses a PN for both inconsistency and incompleteness. This analysis would help improve the software requirements and thus rein the cost and schedule difficulties that would otherwise arise. Hence, this paper proposes algorithms to analyse the PN model of NL requirements for the characteristics of consistency and completeness. Based on the information derived from such analysis, the given requirements are rewritten till they are free of inconsistency and incompleteness.

The proposed approach is demonstrated with a practical example of an Automated Teller Machine (ATM). Additional examples are also included to reinforce the methodology.

The paper structure is described below. Section 2 provides a general idea of the approaches used in the related work. To facilitate further discussion, a short introduction to PNs and basic definitions are given in section 3. The proposed methodology is detailed in section 4, while section 5 illustrates our reference case study of an ATM used to demonstrate the methodology described in section 4. Section 6 is about discussion and validation. Section 7 provides conclusions and future directions for research.

2. Related Work and Research Gap

Several researchers in the literature have tried to formalize the analysis of software requirements for ambiguities, inconsistencies, incompleteness, liveness and reversibility.

Lee, Cha and Kwon [12] define a procedure to convert use cases stated in NL to a CMPN(Constraints-based Modular Petri Nets) model; and define guidelines for finding inconsistency and incompleteness in CMPNs. However, their approach is manual, and the use cases do not conform to the UML. Also, they do not consider alternative/exception flow of use cases.

In [13], Cheung, Cheung and Chow propose a synthesis methodology wherein use cases are specified as labeled PNs or C-nets, and by synthesizing these nets, a system design is derived. They provide an algorithm to verify liveness and reversibility, but their methodology is complex.

In [16], Sinnig, Chalin, and Khendekuse LTS (Labelled Transition Systems) formalize the use case models by a map the use case steps and types. It allows them to detect semantic checks like livelocks and model refinement validation, for which they have developed a tool called "Use Case Model Analyzer".

Zhao and Duan in [10] transform use cases into scenarios and then into their corresponding PNs. These PN models are analyzed, flaws detected and models of the level of Platform Independent Model (PIM) constructed based on analysis. This approach suffers from the need to create intermediate "event frames" for the extraction of objects and messages from each one of the sentence events.

Somè in [14] describes use cases using an abstract syntax (tuple structure) and a concrete one (restricted NL)before mapping them into reactive nets. However, interaction in the case of concurrent use cases is not dealt with.

Sarmiento, Leite and Almentero, in their paper [15], translate Scenarios into equivalent Place / Transition PNs and evaluate them for consistency, correctness and completeness. Based on the analysis of the PNs, they revised the Scenarios. However, their method suffers from the disadvantage that the systems engineer must be knowledgeable in using the syntax and semantic rules described for writing scenarios.

Sarmiento-Calisaya et al. [17] describe a C&L prototype tool that does an analysis of scenarios (Static) and equivalent PNs (Dynamic) for indicators of ambiguity, completeness, and consistency. This transformation part of the tool is, however, sensitive to the correct syntax of scenarios.

Yu et al. [32] use a scenario model based on first-order logic for consistency analysis. However, the approach has been applied only to simple systems and needs further validation on large-scale systems.

From the above background study, we may conclude that the formal methods to detect inconsistency and incompleteness in the literature to date are manual, complex and not based on conventional PNs. In this paper, we have developed a methodology to analyse inconsistency and incompleteness in PNs, thereby in the underlying NL requirements.

3. Background Information and Definitions

Petri nets, also known as Place / Transition Nets, were introduced to model concurrency, non-determinism, and control flow by Carl Adam Petri in 1962. A PN is a group of arcs connecting transitions and places. Places represent the system states, and transitions are the events that occur which may lead to a modification in the states of the system. Places may possess tokens. The tokens enable the transitions when the transition gets fired. The tokens get allocated as per the weight given on arcs. PNs supply a mathematically rigorous modeling framework and are bipartite graphs.[19,20]

3.1. Definition of PNs

A PN is a 5-tuple, [21]

$$PN = (P, T, F, W, M_o) \quad (1)$$

where:

$$P = \{p_1, p_2, \ldots\ldots p_m\} \quad (2)$$

is a finite set of places,

$$T = \{t_1, t_2, \ldots\ldots t_n\} \quad (3)$$

is a finite set of transitions,

$$F \subseteq (P \times T) \cup (T \times P) \quad (4)$$

is a set of arcs (flow relation),

$$W : F \rightarrow (1,2,3 \ldots\ldots) \quad (5)$$

is a weight function,

$$M_o : P \rightarrow (0,1,2,3 \ldots.) \quad (6)$$

is the initial marking,

$$P \cap T = \phi \; and \; P \cup T \neq \phi \quad (7)$$

A PN structure

$$N = (P, T, F, W) \quad (8)$$

without any specific initial marking is denoted by N.

A PN with the given initial marking is denoted by (N, M_o)

3.1.1. Time PNs

Time PN is an extension of the PN model where each transition t_j is associated with two timings $\tau_{1,j}$ and $\tau_{2,j}$.[22] A transition t_j can fire only if it has been enabled for at least time $\tau_{1,j}$, and it must fire before $\tau_{2,j}$ if enabled.[22] Unless otherwise specified, it is assumed

$$\tau_{i,j} = \tau_{2,j} = 0 \quad (9)$$

As a special case if

$$\tau_{i,j} = \tau_{2,j} = t \quad (10)$$

it means that the transition is fired at t if it is enabled.

3.2. Related Properties of PNs

Let

$$PN = (N, M_o) \quad (11)$$

be a system.[23]

A transition t is said to be dead in a PN if no marking of M_0 enables t. A deadlock, or dead marking, is when a marking is not enabling any transition. PN is deadlock-free if no deadlock belongs to M_0; otherwise, it is dead-lockable.

A transition t is live in PN for every marking M in M_0 if there is a marking M′ in M enabling t. PN is said to be live if every transition is live in PN.

A marking M is called a home state of PN if it can be reached from every marking in M_0.

PN is bounded if an integer k exists such that:

$$\forall M \in M_o \quad (12)$$

for each place p,

$$M(p) \leq k \quad (13)$$

A marking M′ is known to be reachable from the marking M if there is a firing sequence σ feasible in (N, M) such that

$$M \overset{\sigma}{\rightarrow} M' \quad (14)$$

Since our primary aim is to develop software requirements which are characterized by consistency and completeness, these terms are also being defined here:

3.2.1. Consistency

Boehm [24] defines a consistent Software Requirements Specification (SRS) as one which does not have conflicting requirements. If a requirement is overridden by another requirement or when users give conflicting requirements, then the requirements are said to be inconsistent.

3.2.2. Completeness

Boehm [24] defines an SRS to be complete when all its parts are present, and each part is fully developed. He opines that no TBDs (TBDs are places in the requirements specification where the decisions are postponed by writing “To be Determined” or “TBD”), no non-existent references, no missing specifications items and no missing functions are the properties that are to be satisfied for completeness to be validated.

4. Proposed Methodology for Analysis

Requirements specifications are used to describe requirements – both functional and non-functional of a system. The functional part covers what the software system should do depending on the system and its relationship with the environment. Non-functional requirements delineate the constraints under which the software system must operate and any design restrictions forced on the system. Here functional requirements alone are being considered.[21]

Researchers have designated the below-given properties to identify and resolve inconsistency and incompleteness.

Inconsistency can be avoided by ensuring the following properties:

- No non-determinism and conflicting requirements, i.e., two transitions should not be enabled simultaneously by a single token in a place [25],
- liveness of the PN, i.e., if life, it indicates the absence of total or partial deadlocks [10,12] and
- no dead transitions.[12]

Proof: Let

$$\gamma = \{\gamma_1, \gamma_2, \ldots\ldots \gamma_k\} \quad (15)$$

where

$$\gamma \in \text{set of conflicting transitions.} \quad (16)$$

Let time

$$t = \{t_1, t_2, \ldots\ldots t_k\} \quad (17)$$

where t_i is the time when a transition can fire after the transition is enabled. Also,

$$\{t_i \neq t_j\} \quad (18)$$

If a place connected to two transitions say γ_i and γ_j, receives a token, transition γ_i is evaluated after

$$t = t_i \quad (19)$$

and γ_j is evaluated after

$$t = t_j \quad (20)$$

and

$$t_i \neq t_j \quad (21)$$

(By assumption) then an inconsistency situation is avoided. Hence our proposition is validated.

In order to solve the inconsistency arising in the case of non-determinism, the “Transitions with time” proposition is being proposed. Accordingly, when two transitions are connected to a place, the transitions do not have any other input place connected to it or any other information on their arcs by assigning one of the transitions a higher priority than the other by introducing a time component.

4.1. Statement of "Transitions with time" proposition

A token at a place, if it enables two transitions simultaneously, then one of the transitions may be assigned a higher priority (by introducing a time component) than the other.

Explanation: An example of two transitions fired simultaneously by a token in a place is shown in Figure 1.

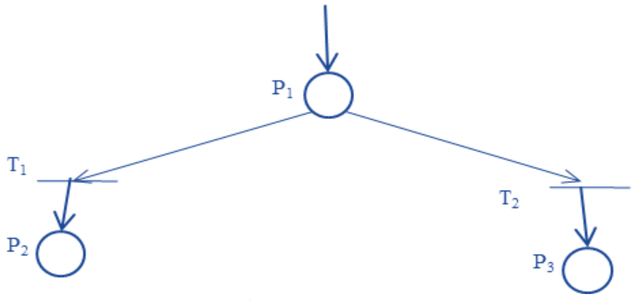

Fig. 1 PN representation of more than one transition connected to a place

By application of the proposed proposition, the modified PN representation is shown in Figure 2, where a time component of

$$t = 0 \quad (22)$$

is specified on the arc connecting place P_1 to transition T_1, and a time component of

$$t = \Delta \quad (23)$$

is specified on the arc connecting place P_1 to transition T_2.

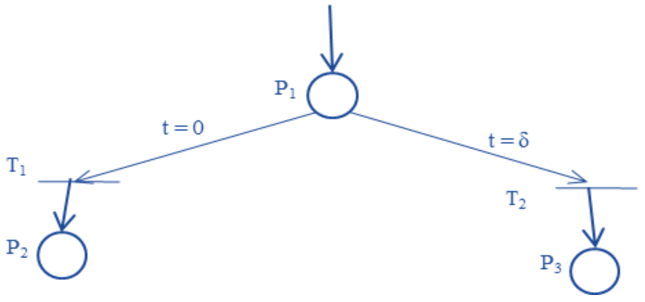

Fig. 2 PN representation of more than one transition connected to a place after application of the "Transitions with time" proposition

To prove the correctness of the proposition, the following two conditions need to be validated:

```
Input: Given a PN.
Output: Identified inconsistencies in the PN.
Result: Inconsistency determined.
Step 1 [Listing all places in the PN.] List all places
        ($P_1$, $P_2$...$P_n$)
that are in the PN representation of the NL requirements;
Step 2 [Verifying non-determinism with Transitions with time proposition.] Simulate the PN;
Traverse the PN;
for each place in the PN do
        if a place is connected to more than one outgoing transition and these transitions do not have any other input place
        connected to it or any information on their arcs then
                return PN is inconsistent;
        end
end
Step 3 [Verifying liveness by simulation.] Simulate the PN;
if simulation is not successful then
        return PN is inconsistent;
end
Step 4 [Verifying that there are no dead transitions in the PN.] Simulate the PN;
Traverse all loops in the PN to determine transitions that are never enabled;
if there exist transitions that are never enabled then
return PN is inconsistent;
end
return PN is consistent;
```

Fig. 3 Algorithm 1 for detecting Inconsistency in a PN

Accumulated tokens due to looping and decision-making structure can affect the subsequent execution of the PN. These tokens have to be removed. This removal can be done through a proposition for handling and removal of residual tokens called the "Residual tokens removal" proposition.[33] It has been included in detail in Appendix 1. This proposition helps in the analysis of PN and the resolution of gaps. Based on the above, an algorithm, as shown in Figure 4, has been developed to analyse the given PN representation of NL requirements for incompleteness.

```
Input: Given a PN.
Output: Identified incompleteness in the PN.
Result: Incompleteness determined.
Step 1 [Listing all transitions in the PN.] List all transitions
                    $(T_1, T_2...T_n)$
and places
                    $(P_1, P_2...P_n)$
that are in the PN representation of the NL requirements;
Step 2 [Functions in requirement specifications in NL.] Obtain list of all functions
                    $(f_1, f_2...f_n)$
from requirements specification in NL;
Step 3 [Verifying no function is absent in the PN.] Simulate the PN;
Traverse the PN and list functions that are not represented by one or more transition in
the PN;
if any function is absent in PN then
          return PN is incomplete;
end
Step 4 [Verifying no non-determinism in the PN.] Obtain reachability report using
PIPE2;
if there exist transitions are not reachable then
return PN is incomplete;
end
Step 5 [Verifying distinct names for all places and transitions in the PN.] Simulate the
PN; Traverse the PN;
if all the places and transitions are not specified by distinct names then
          return PN is incomplete;
end
Step 6 [Verifying no isolated subnets in the PN.] Simulate the PN;
Traverse the PN;
if there exists any transition that remains not traversed then there is an isolated subnet in
the PN then
          return PN is incomplete;
end
Step 7 [Verifying no infinite loops or program abends in the PN.] Simulate the PN;
if there are infinite loops or program abends then
          return PN is incomplete;
end
Step 8 [Verifying no token accumulation in the PN.] Simulate the PN;
if repeated execution of PN results in error then this implies that there could be token
accumulation due to looping and decision-making structure in the PN then
          return PN is incomplete;
end
return PN is complete;
```

Fig. 4 Algorithm 2 for detecting incompleteness in a PN

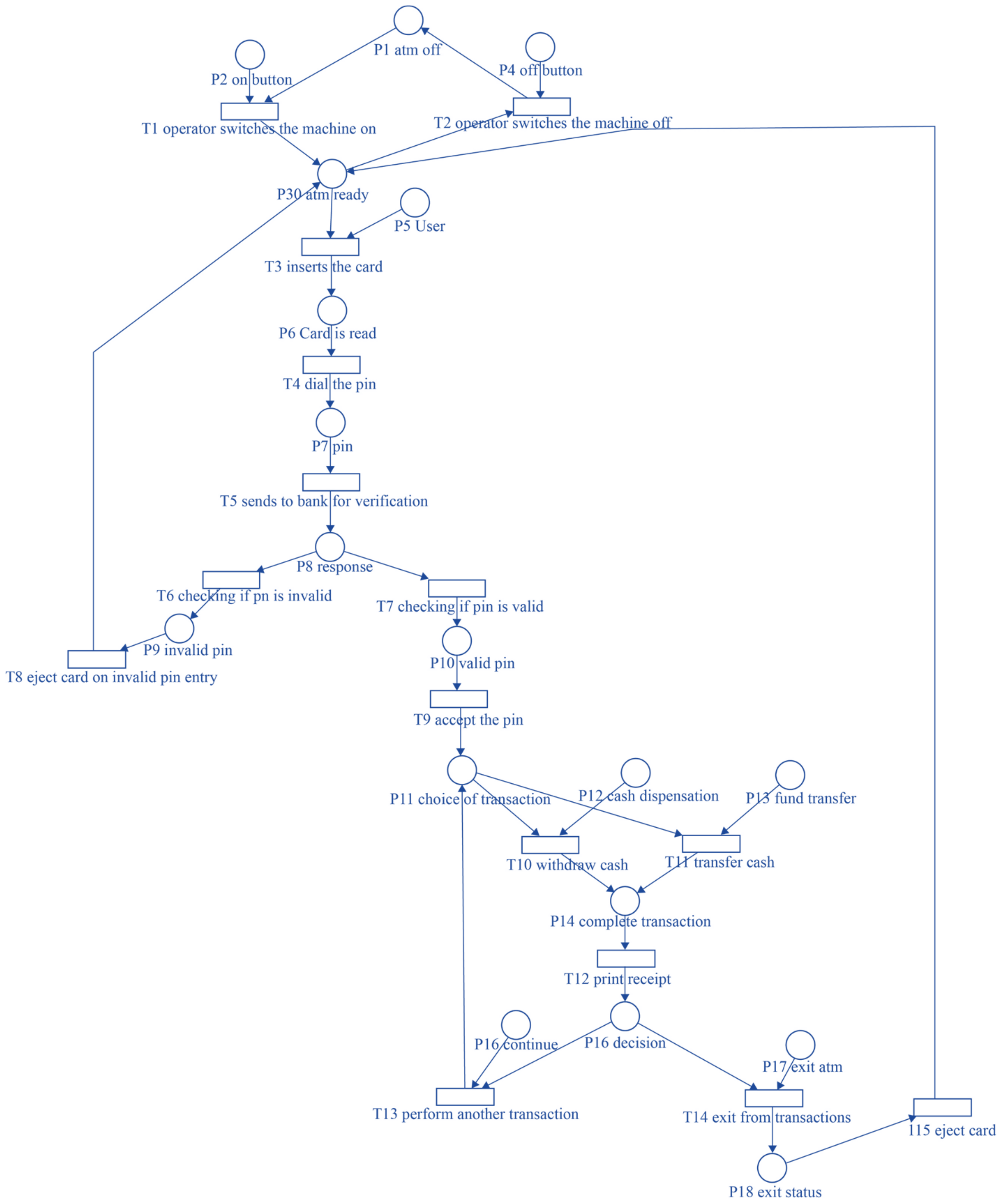

Fig. 5 PN representation of ATM system

The requirements specification and the corresponding PN representation are the inputs to the proposed methodology. From the requirements specification, the list of functions is taken. The PN is simulated using the Renew2.5 tool.[27] Based on algorithm 1 & algorithm 2, analysis is done, and issues are identified. The NL requirements are then rewritten to resolve the identified issues related to inconsistency and incompleteness. This process may have to be iterated if the modified requirements introduce any inconsistency or incompleteness. The requirements are finally approved by the user and then used in the subsequent stages of the SDLC (software development life cycle).

$$M_o(Initial\ Marking) = < p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{17}, p_{18} >< 1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0 > \quad (30)$$

5. Example Case Study – Automated Teller Machine

In order to demonstrate our methodology, a case study of ATMs in their simple form is taken. An ATM's operations were summarized in NL as follows: [28]

"The ATM will read an ATM card, take inputs from a keyboard and then display output, thus interacting with the customer. It then performs the requested action – gives cash or transfers money from one account to another. After this, a printed receipt is produced. This ends one transaction, and the user can perform repeated transactions. The machine can be started or stopped by an operator. The ATM interacts with the bank's computer for validation using the network. The ATM card will be inserted, and a personal identification number (PIN) (this information is sent to the bank for validation) will be entered. If a wrong PIN is entered, a card will be ejected. The customer can perform one or more transactions, after which the card is ejected. The ATM will provide the following services to the customer:

(1) Obtaining Balance Information
(2) Withdrawal of Cash
(3) Transfer of Funds."

The PN for the ATM system taken from the requirements specifications pertaining to the above user requirements is shown in Figure 5 below.

5.1. First iteration – Analysis of PN drawn in Fig 5 using Algorithm 1 and Algorithm 2

Consistency analysis:
Work through Algorithm 1.
Step 1: List of places are:
1. atm off
2. on button
3. atm ready
4. off button
5. user
6. card is read
7. pin
8. response
9. invalid pin
10. valid pin
11. choice of transaction
12. cash dispensation
13. fund transfer
14. complete transaction
15. decision
16. continue
17. exit atm
18. exit status

Step 2: A place connected to more than one outgoing transition and where these transitions do not have any other input place connected to it or any information on their arcs is "response" (P_8) place and transitions checking if the pin is invalid (T_6) &checking if the pin is valid (T_7) as seen in Fig. 5. The PN drawn in Fig. 5 indicates that the NL requirements are inconsistent.

Step 3: Simulate the PN to verify liveness. Simulation is successful, and hence PN is consistent with regard to liveness.

Step 4: Simulate the PN to verify that there are no dead transitions. Traverse all loops in the PN to determine transitions that are never enabled. There exist no transitions that are never enabled, and hence PN is consistent with regard to dead transitions.

5.1.1. Completeness Analysis

Work through Algorithm 2.
Corrected list of Transitions
Step 1: List of transitions are:
1. operator switches the machine on
2. operator switches the machine off
3. inserts the card
4. dial the pin
5. sends to the bank for verification
6. checking if the pin is invalid
7. checking if the pin is valid
8. eject card on invalid pin entry
9. accept the pin
10. withdraw cash
11. transfer cash
12. print the receipt
13. perform another transaction
14. exit from transactions
15. eject card.

The list of places is the same as given previously.
Step 2: Obtain the list of all functions.

1. switching on and off the atm
2. verify the validity of the pin

3. eject card
4. display balance information
5. withdrawal of cash
6. cash transfer
7. print the receipt
8. performing transactions
9. exit from the transaction

Step 3: Traverse the PN and list functions absent in PN. One function, namely 4 from step 2 above, is absent; hence, PN is incomplete with regard to functions. A transition display of balance information(T_{12}) is added to represent the missing functionality in the PN in Fig. 6.

Step 4: Obtained reachability report using PIPE2 to determine whether there is non-determinism in the PN. There exist no transitions that are not reachable. PN is complete with regard to non-determinism.

Step 5: Traverse PN. All places and transitions in the PN have distinct names. PN is complete with regard to distinct names.

Step 6: Traverse PN. There are no isolated subnets. PN is complete with regard to isolated subnets.

Step 7: Simulate PN to determine if there are infinite loops or program abends.

Step 8: Simulate PN. There is no token accumulation. PN is complete with regard to token accumulation.

Thus, the PN drawn in Fig. 5 is inconsistent and incomplete; therefore, the NL requirements need to be modified.

5.1.2. Observations and Solution

Inconsistency

From the application of algorithm 1 on the PN drawn in Fig. 5, it is seen that the two transitions viz. T_6 and T_7 are enabled together. Hence it is a case of inconsistency. To resolve this inconsistency problem, different priorities are to be assigned for T_6 and T_7 as per the 'Transitions with time' proposition. In this case, we assign T_6 a higher priority than T_7. This resolves the inconsistency in the PN. If the functional condition of T_6 is enabled, then transition T_6 is fired. Otherwise, after a duration $\Delta 1$ transition T_7 is fired.

Incompleteness

(1) During simulation, the PN execution results in an endless looping condition. It is seen that the requirements of the following functionalities - the choice to repeat the transaction, accept the pin given, validate the pin and eject the card on the wrong pin are inadequately given, which leads to the infinite looping situation.

(2) A single user repeatedly entering the pin for validation would prevent others from using the atm. A threshold on the number of times a user is allowed to enter the pin for validation must be incorporated. Alternatively, a time limit for the action can be incorporated.

(3) A single user repeatedly performing transactions preventing/delaying other users. Again, either a limit on the number of transactions a user can be allowed must be incorporated, or a time limit for the user usage can be incorporated.

To satisfy completeness, the following needs to be done:

To do (1), a limit on the number of attempts for dialling the pin, say three times, is added as shown in PN drawn in Fig. 6. A maximum of three attempts for dialling the pin has been introduced by adding a transition repeat pin entry (T_8).

For (2), a time limit within which the user must take action while making transactions, say thirty seconds, must be introduced. In Fig. 6 drawn, a time limit for executing a transaction has been introduced by adding two transitions, activating the timer (T_{11}) and exceeding the time limit (T_{15}) and two places, "timer request" (P_{11}) & "end of the timer" (P_{13}) with a time limit (t3) of 30 seconds.

Though identified, the maximum number of transactions and a timer for dialling the pin have not been included, as the repetition in the PN would not add to the analysis.

5.2. Modified requirements for Automated Teller Machine

Based on the analysis, the requirements are rewritten to resolve the identified inconsistencies and incompleteness as below:

"The ATM will be able to read an ATM card, take inputs from a keyboard and then display output, thus interacting with the customer. It then performs the requested action – gives balance information, allows cash withdrawal, or transfers money from one account to another. After this, a printed receipt is produced. This ends one transaction, and the user can perform repeated transactions. The machine can be started or stopped by an operator. The ATM interacts with the bank's computer for validation using the network. The ATM card will be inserted, and a personal identification number (PIN) (this information is sent to the bank for validation) will be entered. If a wrong PIN is entered, a card will be ejected. The customer is allowed three attempts to enter the correct PIN; else card is ejected out. Similarly, the customer is allowed to perform a maximum of three transactions, after which the card is ejected out. Time taken by the customer for PIN entry or for making transactions needs to be limited to 30 seconds in each case. In case of delay by the customer, the card should be ejected out."

Note 1: The bold-faced text indicates the changes made in the requirements.

The PN of the ATM system is then redrawn based on the rewritten requirements. The PN for the above specifications drawn using our approach is depicted in Fig. 6.

$$M_o(Initial\ Marking) = < p_1, p_2, p_3, p_4, p_5, p_6, p_7, p_8, p_9, p_{10}, p_{11}, p_{12}, p_{13}, p_{14}, p_{15}, p_{16}, p_{17}, p_{18}, p_{19}, p_{20}, p_{21} >< 0, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0 > (31)$$

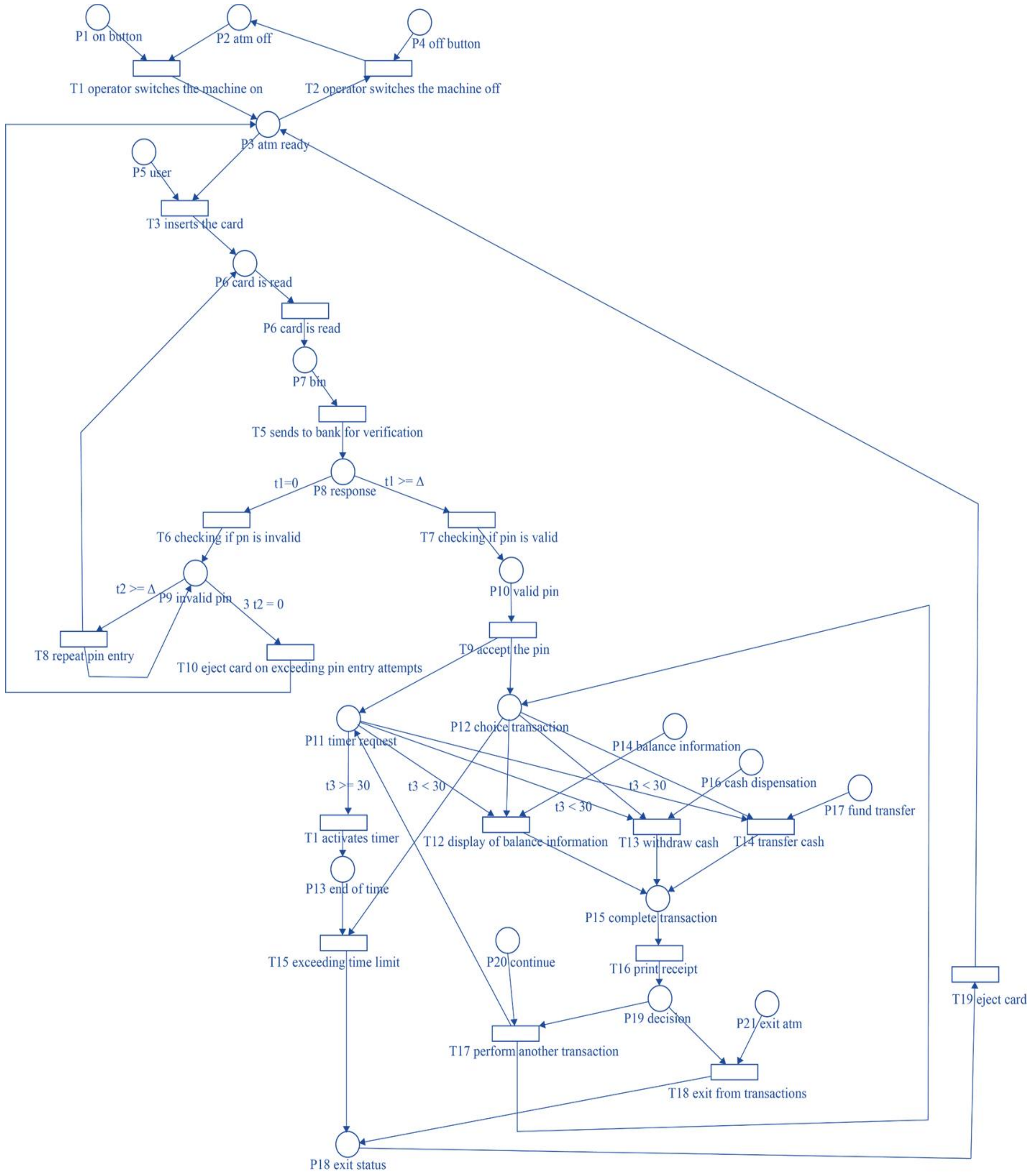

Fig. 6 PN representation of ATM system after the first iteration of application of Algorithm 1 and Algorithm 2.

One transition - eject card on invalid pin entry in Fig. 5 is removed, and five transitions - repeat pin entry, eject card on exceeding pin entry attempts, activates timer, display of balance information and exceeding time limit have been added in Fig. 6.

5.3. Second iteration

Analysis of PN drawn in Fig 6 using Algorithm 1 and Algorithm 2

Now the methodology is iterated.

Consistency analysis:

Application of Algorithm 1 shows that the PN is consistent.

Completeness analysis:

Work through Algorithm 2.

Step 1: List of transitions
1. operator switches the machine on
2. operator switches the machine off
3. inserts the card
4. dial the pin
5. sends to the bank for verification
6. checking if the pin is invalid
7. checking if the pin is valid
8. repeat pin entry
9. accept the pin
10. eject card on exceeding pin entry attempts
11. activates timer
12. display of balance information
13. withdraw cash
14. transfer cash
15. exceeding time limit
16. print receipt
17. perform another transaction
18. exit from transactions
19. eject card.

The list of places are:
1. atm off
2. on button
3. atm ready
4. off button
5. user
6. card is read
7. pin
8. response
9. invalid pin
10. valid pin
11. timer request
12. choice of transaction
13. end of timer
14. balance information
15. complete transaction
16. cash dispensation
17. fund transfer
18. exit status
19. decision
20. continue
21. exit ATM

Step 2: Obtain the list of all functions.
1. switching on and off the atm
2. verify the validity of the pin
3. repeat pin entry
4: eject card on exceeding pin entry attempts
5: activates the timer
6: exceeding the time limit
7: the display of balance information
8: withdrawal of cash
9. cash transfer
10. print receipt
11. performing transactions
12. exit from the transaction
13. eject card

Step 3: Traverse the PN and list functions absent in PN. All the functions are present. PN is complete with regard to functions.

Step 4: Obtained reachability report using PIPE2 to determine whether there is non-determinism in the PN. There exist no transitions that are not reachable. PN is complete with regard to non-determinism.

Step 5: Traverse PN. All places and transitions in the PN have distinct names. PN is complete with regard to distinct names.

Step 6: Traverse PN. There are no isolated subnets. PN is complete with regard to isolated subnets.

Step 7: Simulate PN to determine if there are infinite loops or program abends. There is neither any infinite loop nor any program abend. PN is complete with regard to infinite loops or program abends.

Step 8: Simulate PN. Two errors are identified.

5.3.1. Observations and Solution

Inconsistency

There is no inconsistency.

Incompleteness

a) In the simulation of the PN, it is noticed that there is an accumulation of tokens in the “invalid pin” (P_9) place which affects the repeated execution of the PN. This happens

(1) When the user enters the pin correctly at the third instance after entering it wrongly the first two times, two tokens get collected in P_9 place.

(2) When the user enters the pin correctly at the second instance after entering it wrongly the first time, one token gets collected in P_9 place.

An additional transition emptying of tokens (T_9) and two additional places, "unused token" (P_{10}) and "count of invalid pin entries" (P_{12}), are added to the PN to ensure and correct the flow, usage and removal of tokens such that the functionality given in the specification can be completed successfully. For this processing of residual tokens, the "Residual tokens removal" proposition in [22] has been used.

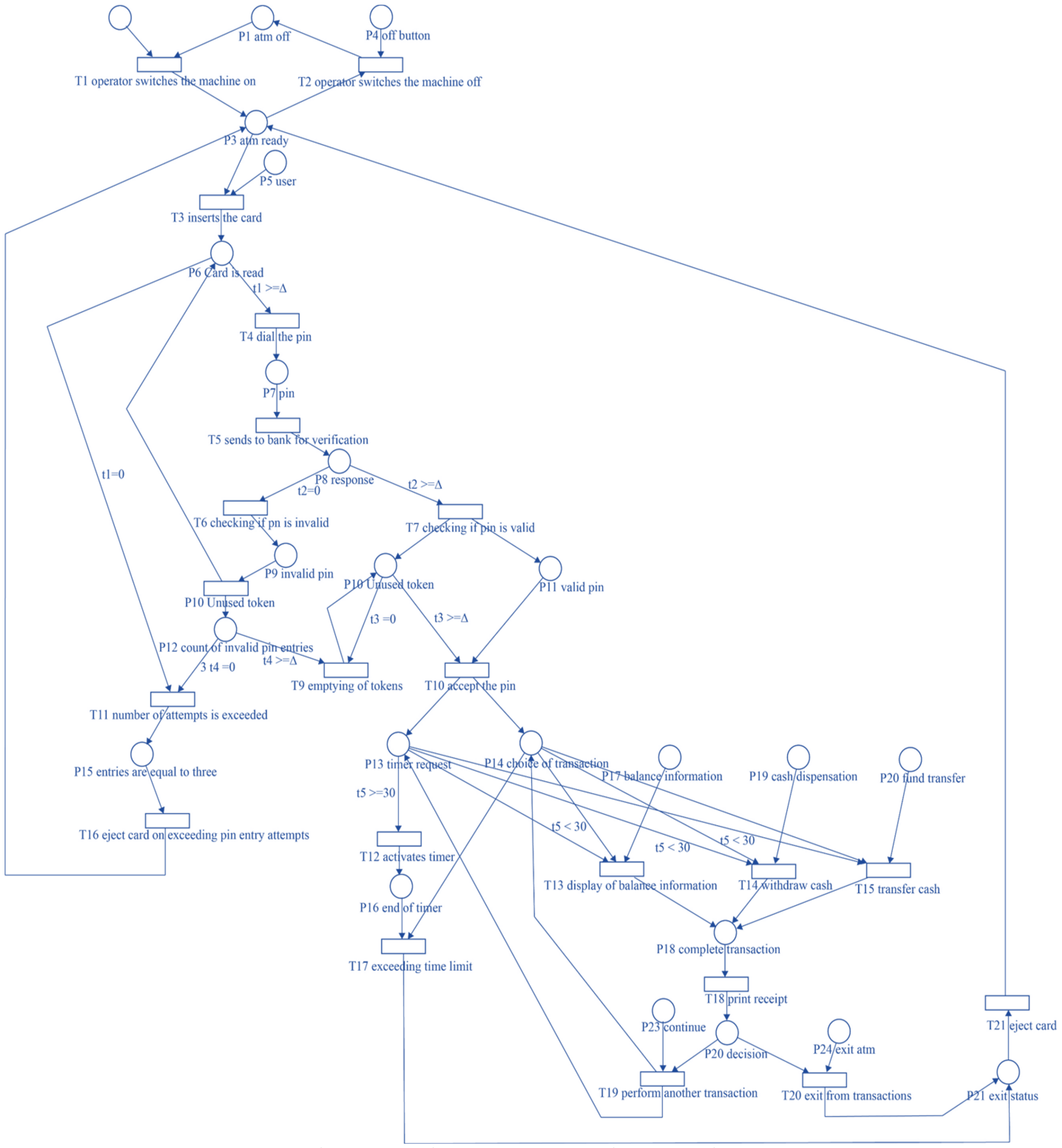

Fig. 7 PN representation of ATM system after the second iteration of application of Algorithm 1 and Algorithm 2.

b) Also, there is the accumulation of a token in the "card is read" (P_6) place due to the token from transition repeat pin entry (T_8) which goes back to place P_6 in the situation when T_8 is fired thrice, i.e., a wrong pin is entered thrice. To remove this leftover token, P_6 is made to fire two transitions - the number of attempts is exceeded (T_{11}) at

$$t_1 = 0 \quad (32)$$

and dial the pin (T_4) at

$$t_1 \geq \Delta \quad (33)$$

PN is incomplete with regard to the accumulation of tokens. Thus, the PN drawn in Fig. 6 is consistent but incomplete. Though the NL requirements are correct, the PN implemented needs to be modified. Two transitions - the number of attempts is exceeded&emptying of tokens have been added in Figure 7.

The corrected PN representation is depicted in Fig. 7.

5.4. Proof of Consistency and Completeness of the final Petri net model

The ATM PN (Fig. 7) is now examined for consistency and completeness. The property of consistency and completeness of the modified ATM PN is verified by running algorithm 1 and algorithm 2, respectively. By simulation of the PN, the execution of various transitions is found to be successful. Thus, feedback obtained in the two iterations was used to improve the requirements by removal of inconsistency and incompleteness. It can be seen from this example that the proposed algorithms are easy and are not time-consuming or laborious to apply.

6. Discussion and Validation

The ATM case study demonstrates the use of our technique for a typical application of an ATM. Similarly, three other case studies (Table 1) have been carried out by using NL requirements for the patient monitoring system [29], Assembly system [30] and Library system.[31] All these cases have been investigated using the methodology described in this paper. In all the above cases, PN representations of the requirements have been analysed and requirements corrected, proving the soundness of our proposed methodology. The detailed analyses of the three case studies are included in Appendix 2. A snapshot of the outcomes of the case studies is presented in the table below.

Table 1. Results of analysis of consistency and completeness of case studies (requirements)

	Patient monitoring system	**Assembly system**	**Library System**
Inconsistent transitions identified from consistency analysis	Two	Zero	Eight
New transitions identified from completeness analysis	Three	Five	Two
New functionalities identified from completeness analysis	Two	Three	Two

Many researchers in literature have attempted to formalize software requirements analysis to free them of ambiguities, inconsistencies, incompleteness, etc., by converting them to PNs. Using PNs for analysis helps us greatly improve the requirements specifications. The methodology presented here incorporates analysis and feedback and does not need specific know-how to use the approach. Therefore, the usage of this method is highly advantageous. The method proposed in this paper for the analysis of PNs is compared with the methods proposed by Zhao and Duan [10], Lee et al. [12] and Sarmiento et al. [15] in table 2 below.

Table 2. Comparison of the different methods for analysing PN representation of software requirements.

	Methodology	**Analysis was done for**	**Specific know-how required / Disadvantages / Advantages**
Zhao and Duan [10]	(1) Use cases are transformed into Scenarios. (2) Scenarios are transformed into Timed and Controlled PNs (TCPN). (3) The TCPNs are analyzed using a PN tool (PIPE2). (4) Feedback is used to rebuild the scenarios and use case descriptions. (5) After correction, Platform Independent Models (PIM) are built.	(1) Completeness. (2) Correctness. (3) Consistency.	(1) Complex industrial systems are difficult to express as PNs. (2) Intermediate "event frames" for each of the sentence events are needed.
Lee et al. [12]	(1) From use, cases create an action-condition table with event names and pre- and post-conditions.	(1) Completeness. (2) Consistency.	(1) Intermediate models are created.

	(2) Convert the action-condition tables into Constraints-based Modular Petri Nets (CMPNs). (3) Guidelines were developed to analyze the CMPNs for consistency and completeness.		(2) Alternative/exception flows of use cases are not considered. (3) The use cases do not conform to UML.
Sarmiento et al. [15]	(1) Define a Scenario language. (2) Define mapping rules from Scenarios to PNs. (3) Transform Scenarios to PNs. (4) Integrate partial PNs into an integrated PN. (5) Using reachability analysis, the PN is analyzed, and the scenarios are revised.	(1) Correctness. (2) Consistency. (3) Completeness.	(1) Good knowledge of writing scenarios is needed.
Proposed Approach	(1) Use algorithm 1 for consistency analysis. (2) Use algorithm 2 for completeness analysis. (3) Note the identified inconsistencies and incompleteness to revise and modify the given requirements from which the PN was developed. (4) Repeat the analysis and modify the requirements if needed. (5) Use improved/corrected requirements for the next steps in SDLC.	(1) Consistency. (2) Completeness.	(1) Both Consistency and Completeness are identified. (2) Can be used for conventional PNs. (3) Semi-automated tool available to generate PNs easily from NL requirements. (4) No specific knowledge is required to use the algorithms presented.

7. Conclusion and Future Course of Work

Proper requirements specification is key to the successful completion of software projects. Petri nets are a convenient approach for representing, analysing and correcting deficiencies in the software requirements. In this paper, assuming the requirements and their PN expressions are given, an analysis of the PNs has been done to identify inconsistency and incompleteness in requirements. The feedback is used to improve the quality of the software requirements.

The methodology of improving the requirements specification represented as Petri nets is demonstrated in this paper by using the Automated Teller Machine example. By this methodology, NL requirements represented by the PNs have undergone modifications and improved considerably. Applying this step before the next stage in the software development life cycle would help reduce error propagation and consequent cost and schedule overruns. The proposed methodology has a strong theoretical foundation. The algorithms and conditions of proof given are simple and easy to apply. Though not applicable to conventional NL syntax directly, it is useful once the NL specifications are translated to Petri nets (an automated method is proposed for converting natural language requirements into Petri nets in [33]).

The technique needs to be verified on sizeable and complex systems. In future, techniques for analyzing other errors in requirements like ambiguity, repetition, conflicts, etc., would form our scope of research. Also, we would like to research preventing state space explosion, which usually occurs when PNs are used for complex systems.

Appendix 1

"Residual tokens removal" proposition: In an iterative decision-making loop, when the decision becomes true at the nth iteration, there will be an accumulation of tokens due to the false conditions which have occurred (n – 1) times. These (n – 1) tokens are to be removed so that the process continues unhindered at the nth iteration when the decision becomes true. Normally PNs do not take care of such situations. We expect decision-making loops to occur in our study; therefore, we give the following proposition. This proposition plays an important role in our approach discussed in section 4.

Statement of the proposition: A loop of one additional transition and two additional places can clear accumulated tokens resulting from the iterative decision-making loop.

Explanation: An example of an iterative decision-making loop is shown in Figure A.1.1 In this representation, it is seen that when a choice needs to be made between true and false values of the password, one of the two transitions is fired to test the password value. The transition check if the password is invalid (T_2) that checks if the password is a false value occurs first at the time $t = 0$ (34)
and the transition check if the password is valid (T_3) that checks if it is a true value occurs at the time $t \geq \Delta$ (35)

In the situation when the value tests false and there is an iterative decision-making loop with the maximum number of iterations being three, the invalidated place (here "password invalid" (P_3) place) accumulates tokens which do not get automatically cleared. These accumulated tokens need to be removed for the system's proper functioning.

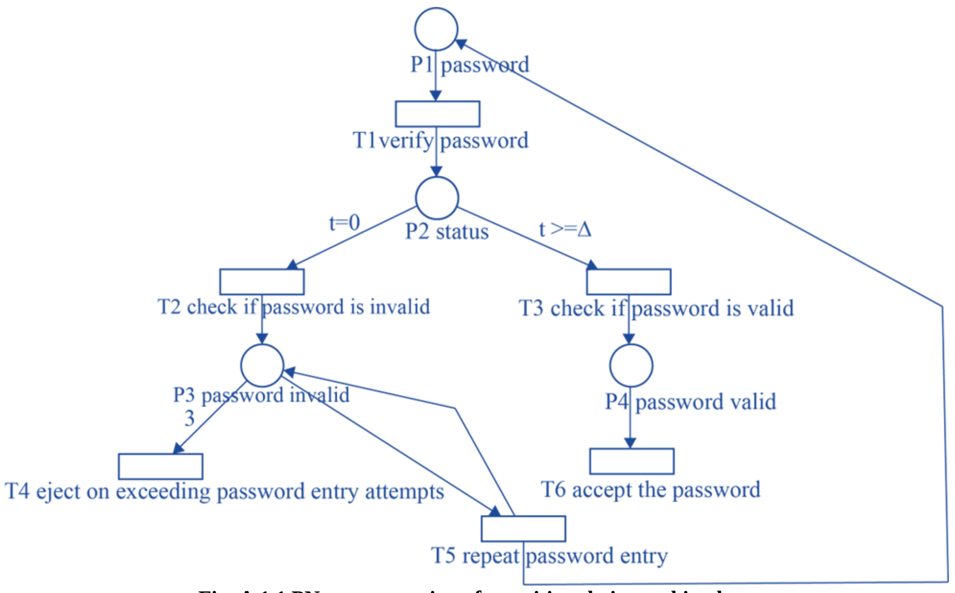

Fig. A.1.1 PN representation of repetitive choice-making loop

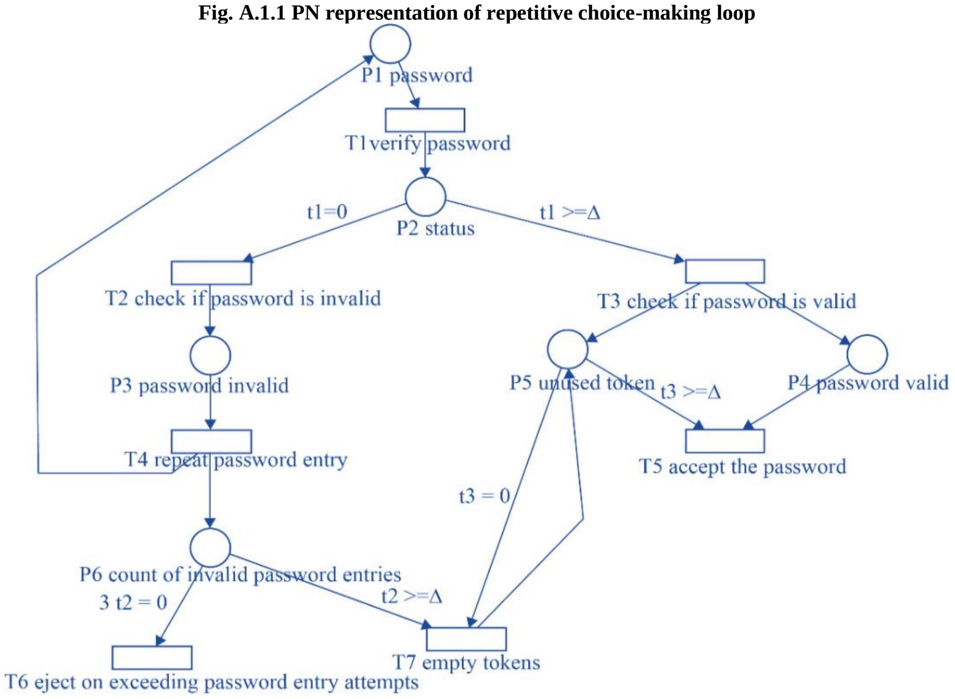

Fig. A.1.2 PN representation of repetitive choice-making loop after application of "Residual tokens removal" proposition.

By application of our proposition, the modified PN representation is shown in Figure A.1.2, where an additional transition empty tokens (T_7) and two additional places, namely "unused token" (P_5) and "count of invalid password entries" (P_6), are introduced. To prove the correctness of our proposition, the following two conditions need to be validated.

Condition 1: When the decision is false, the P_6 place should receive a token and P_5 and "password valid" (P_4) places should not receive any token.

Condition 2: If the decision is true at the nth iteration, P_5 and P_4 places should receive a token each, and place P_6 should clear the token accumulated in the previous iterations.

Proof of condition 1: Firing of transition T_2 (implies) P_6 is incremented by one token; P_4 and P_5 receive no token.

Proof of condition 2: Firing of transition T_3 (implies) P_4 and P_5 receive a token each, P_6 does not get incremented, and accumulated tokens at P_6 get cleared by firing of the empty transition tokens (T_7) repetitively.

Hence our proposition is validated.

Appendix 2

Case Study – Patient Monitoring System

Requirements:

“1) A patient monitoring program is required for a hospital.

2) Each patient is monitored by an analog device which measures factors such as pulse, temperature, blood pressure, and skin resistance. 3) The program reads these factors on a periodic basis (specified for each patient) and stores these factors in a database. 4) For each patient, safe ranges for each factor are specified (e.g., patient X's valid temperature 'range is 98 to 99.5 degrees Fahrenheit). 5) If a factor falls outside of the patient's safe range, or if an analog device fails, the nurse's station is notified.” [29]

Assumptions

The below-given assumptions have been made while developing the PN as the required information has not been specified in the initial specifications:

1. Setup and change of period of measurement of factors are not included.

2. Setup and maintenance of the database needed for the system are not included.

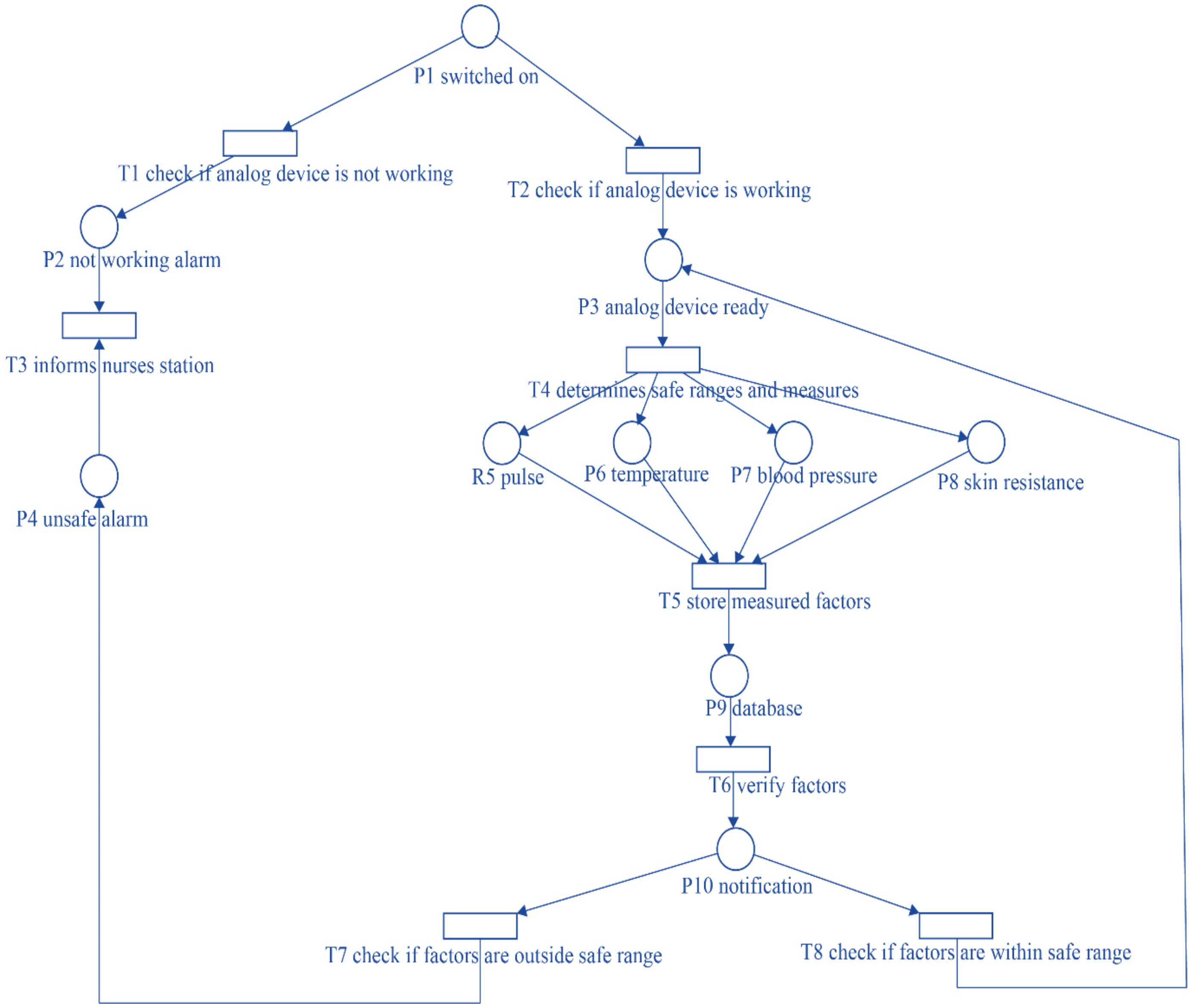
P1 switched on
T1 check if analog device is not working
T2 check if analog device is working
P2 not working alarm
P3 analog device ready
T3 informs nurses station
T4 determines safe ranges and measures
R5 pulse
P6 temperature
P7 blood pressure
P8 skin resistance
P4 unsafe alarm
T5 store measured factors
P9 database
T6 verify factors
P10 notification
T7 check if factors are outside safe range
T8 check if factors are within safe range

Fig. A.2.1. PN representation of patient monitoring system.

Analysis for detection of Consistency and Completeness of PN

First iteration - Analysis of PN drawn in Fig. A.2.1 using Algorithm 1 and Algorithm 2

Consistency analysis:
Work through Algorithm 1.

Step 1: List of places are:
1. switched on
2. not working alarm
3. analog device ready
4. unsafe alarm
5. pulse
6. temperature
7. blood pressure
8. skin resistance
9. database
10. notification

Step 2: A place connected to more than one outgoing transition and where these transitions do not have any other input place connected to it or any information on their arcs is “notification” (P_{10}) place and transitions check if factors are outside safe range (T_7) &check if factors are within a safe range (T_8) as seen in Fig. 1.

The PN drawn in Fig. A.2.1 indicates inconsistent NL requirements.

Solution: From the application of algorithm 1 on the PN drawn in Fig. 1, it is seen that the two transitions, viz. check if factors are outside the safe range (T_7) and check if factors are within the safe range (T_8), are enabled together. Hence it is a case of inconsistency. To resolve this inconsistency problem, different priorities are assigned for T_7 and T_8 as per the “Transitions with time” proposition. In this case, we assign T_7 a higher priority than T_8. This resolves the inconsistency in the PN. If the functional condition of T_7 is enabled, transition T_7 is fired. Otherwise, after a duration Δ transition T_7 is fired.

Step 3: Simulate the PN to verify liveness. Simulation is successful, and hence PN is consistent with regard to liveness.

Step 4: Simulate the PN to verify that there are no dead transitions. Traverse all loops in the PN to determine transitions that are never enabled. There exist no transitions that are never enabled, and hence PN is consistent with regard to dead transitions.

Completeness analysis:
Work through Algorithm 2.

List of Transitions

Step 1: List of transitions are:
1. check if the analog device is not working
2. check if the analog device is working
3. informs nurses' station
4. determines safe ranges and measures
5. store measured factors
6. verify factors
7. check if factors are outside the safe range
8. check if factors are within the safe range

The list of places are:
1. switched on
2. not working alarm
3. analog device ready
4. unsafe alarm
5. pulse
6. temperature
7. blood pressure
8. skin resistance
9. database
10. notification

Step 2: Obtain a list of all functions from the requirements specification (availability of a complete requirements specification document is assumed at this point).
1. Turning the device on and off
2. Check if the analog device is working
3. Inform the nurse's station
4. Determine safe ranges and measures
5. Store-measured factors
6. Verify factors
7. check if factors are within a safe range

Step 3: Traverse the PN and list functions absent in PN. One function, namely function 1 from step 2 above, is absent; hence, PN is incomplete.

Step 4: Obtained reachability report using PIPE2 to determine whether there is non-determinism in the PN. There exist no transitions that are not reachable. PN is complete with regard to non-determinism.

Step 5: Traverse PN. All places and transitions in the PN have distinct names. PN is complete with regard to distinct names.

Step 6: Traverse PN. There are no isolated subnets. PN is complete with regard to isolated subnets.

Step 7: Simulate PN to determine if there are infinite loops or program abends. The PN executes in an infinite loop. Since the working of the patient monitoring system is 24*7, this is not considered an error. However, this indicates that start and stop functionality are not defined. Since this is already determined by Step 3 above, we consider that the PN is complete with regard to infinite loops. The PN executes without abending.

Step 8: Simulate PN. There is no token accumulation. PN is complete with regard to token accumulation.

Modified NL requirements for patient monitoring system

Based on the analysis, the requirements are rewritten to resolve the identified inconsistencies and incompleteness as below:

"1. A hospital needs a patient monitoring program for its patients
2. Setup of safe ranges for each measured factor is done for the patient in the monitoring program.
3. Patients are measured by the analog device.
4. If the analog device is off, it is switched on before use. In the end, after use, the analog device is switched off.
5. The analog device measures the following factors – pulse, temperature, blood pressure, and skin resistance.
6. The program monitors the factors.
7. The values of the factors are stored in a database.
8. The nurse station is informed if the factors' values of a patient fall outside the safe range.
9. The nurse station is also informed if the analog device fails to measure one of the following factors – pulse, temperature, blood pressure and skin resistance."

Note 3: The bold-faced text indicates the changes made in the requirements.

Three transitions - turn the device on, turn the device off and check analog device status have been added in Fig. 2.

Proof of Consistency and Completeness of the final PN model

The Patient monitoring system PN (Fig. A.2.2) is now examined for consistency and completeness.

The property of consistency and completeness of the modified patient monitoring system PN is verified by running algorithm 1 and algorithm 2, respectively. By simulation of the PN, the execution of various transitions is found to be successful. The system is now consistent and complete.

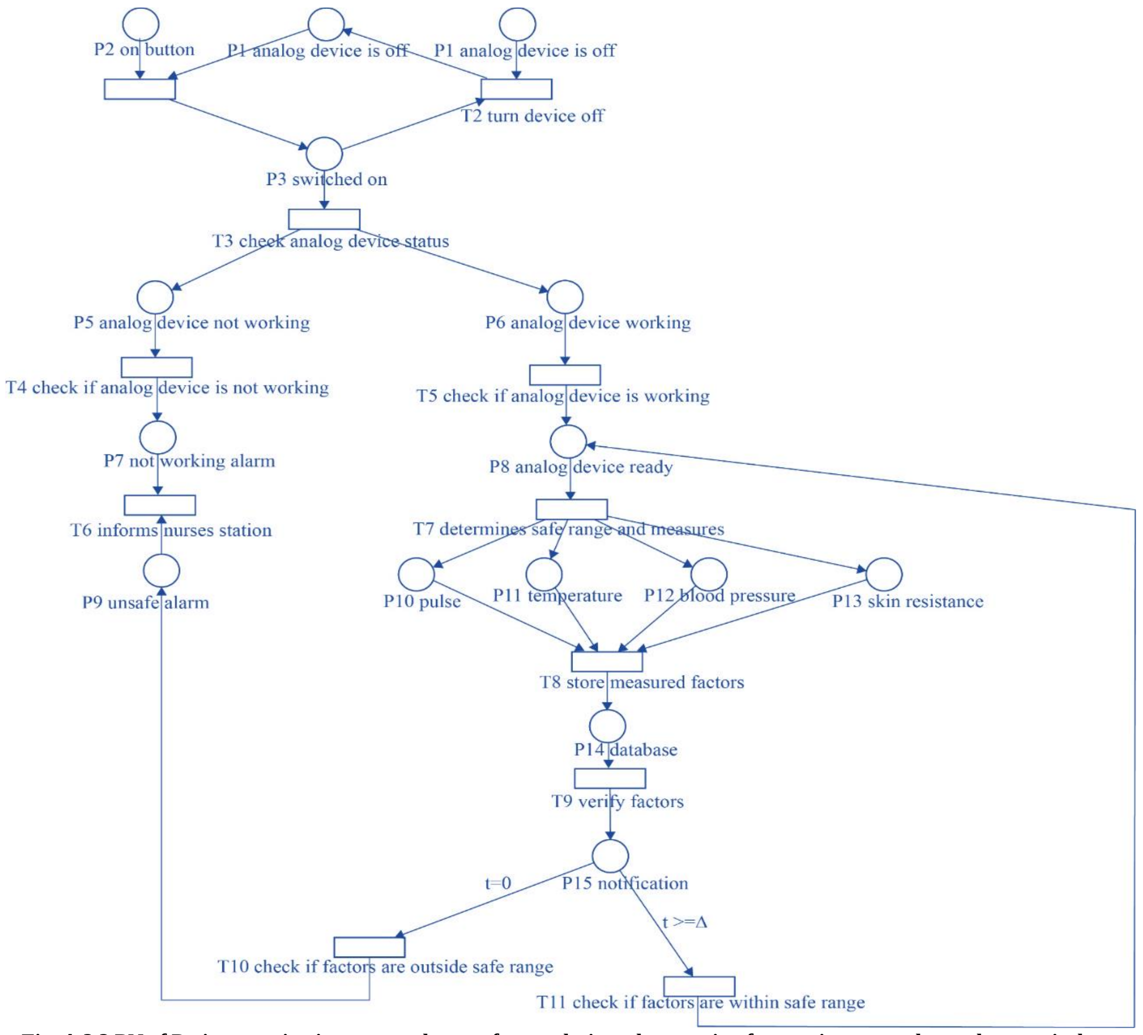
P2 on button
P1 analog device is off
P1 analog device is off
T2 turn device off
P3 switched on
T3 check analog device status
P5 analog device not working
P6 analog device working
T4 check if analog device is not working
T5 check if analog device is working
P7 not working alarm
P8 analog device ready
T6 informs nurses station
T7 determines safe range and measures
P9 unsafe alarm
P10 pulse
P11 temperature
P12 blood pressure
P13 skin resistance
T8 store measured factors
P14 database
T9 verify factors
t=0
P15 notification
t >=Δ
T10 check if factors are outside safe range
T11 check if factors are within safe range

Fig. A.2.2 PN of Patient monitoring system drawn after analysis and correction for consistency and completeness is done.

Case study – Assembly system
Requirements Specifications

"An assembly unit consists of a user, a belt, a vision system, a robot with two arms, and a tray for assembly. The user puts the dish and cup onto the belt, and the belt conveys the parts towards the vision system. The vision system senses on entry of a part into the sensor zone and informs the belt to stop. The vision system then recognizes the type of part and informs the robot to pick it up from the belt. The robot picks, and the belt moves. An assembly is complete when a dish and cup are placed on the tray separately by the arms of the robot."[30]

Assumptions

The below-given assumption has been made while developing the PN as the required information has not been specified in the initial specifications:

1) Assumed that robot uses one arm to pick a dish and another arm to pick a cup.

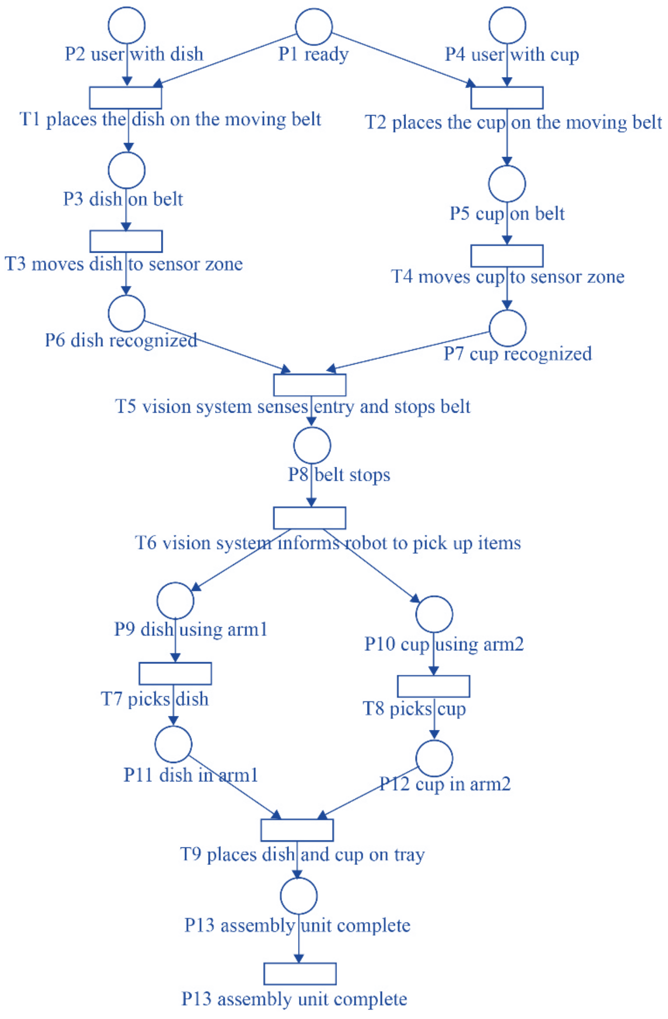
P2 user with dish
P1 ready
P4 user with cup
T1 places the dish on the moving belt
T2 places the cup on the moving belt
P3 dish on belt
P5 cup on belt
T3 moves dish to sensor zone
T4 moves cup to sensor zone
P6 dish recognized
P7 cup recognized
T5 vision system senses entry and stops belt
P8 belt stops
T6 vision system informs robot to pick up items
P9 dish using arm1
P10 cup using arm2
T7 picks dish
T8 picks cup
P11 dish in arm1
P12 cup in arm2
T9 places dish and cup on tray
P13 assembly unit complete
P13 assembly unit complete

Figure A.2.3 PN representation of Assembly system.

Analysis for detection of Consistency andCompleteness of PN

First iteration - Analysis of PN drawn in Fig. A.2.3 using Algorithm 1 and Algorithm 2

Consistency analysis:

Work through Algorithm 1.
Step 1: List of places are:
1) ready
2) user with dish
3) user with a cup
4) dish on the belt
5) cup on the belt
6) dish recognized
7) cup recognized
8) belt stops
9) dish using arm1
10) cup using arm2
11) dish in arm1
12) cup in arm2
13) assembly unit complete

Step 2: There is only one place connected to more than one outgoing transition. However, these transitions have other input places connected to them.

Place "ready" (P_1) is connected to two transitions placing the dish on the moving belt (T_1) and placing the cup on the moving belt (T_2). However, these two transitions each have another input place connected, i.e., "user with the dish" (P_2) and "user with cup" (P_4), respectively.

PN is consistent with regard to non-determinism with the "Transitions with time" proposition.

Step 3: Simulate the PN to verify liveness. Simulation is successful, and hence PN is consistent with regard to liveness.

Step 4: Simulate the PN to verify that there are no dead transitions. Traverse all loops in the PN to determine transitions that are never enabled. There exist no transitions that are never enabled, and hence PN is consistent with regard to dead transitions.

Completeness analysis:

Work through Algorithm 2.
Corrected list of Transitions
Step 1: List of transitions are:
1) places the dish on the moving belt
2) places the cup on the moving belt
3) moves dish to sensor zone
4) moves up to the sensor zone
5) vision system senses entry and stops the belt
6) vision system informs the robot to pick up items
7) picks a dish
8) picks the cup
9) places dish and cup on the tray
10) senses task done and informs belt to move

The list of places are:
1) ready
2) user with dish
3) user with a cup
4) dish on the belt
5) cup on the belt
6) dish recognized
7) cup recognized
8) belt stops
9) dish using arm1
10) cup using arm2
11) dish in arm1
12) cup in arm2
13) assembly unit complete

Step 2: Obtain the list of all functions.
1) switch on and switch off the assembly system
2) placing the dish and cup on the moving belt
3) moving dish and cup to sensor zone
4) vision system sensing entry and stopping the belt
5) vision system informing robot to pick up items
6) picking dish and cup
7) placing dish and cup on the tray
8) sensing task done and informing belt to move
9) repeating assembly of dish and cup

Step 3: Traverse the PN and list functions absent in PN. One function, namely function 1 from step 2 above, is absent; hence, PN is incomplete.
Step 4: Obtained reachability report using PIPE2 to determine whether there is non-determinism in the PN. There exist no transitions that are not reachable. PN is complete with regard to non-determinism.
Step 5: Traverse PN. All places and transitions in the PN have distinct names. PN is complete with regard to distinct names.
Step 6: Traverse PN. There are no isolated subnets. PN is complete with regard to isolated subnets.
Step 7: Simulate PN. The PN executes in an infinite loop. This indicates that start and stop functionality is not defined. Since this is already determined by Step 3 above, we consider that the PN is complete with regard to infinite loops. The PN executes without abending.
Step 8: Simulate PN. There is no token accumulation. PN is complete with regard to token accumulation.

Thus, the PN drawn in Fig. A.2.3 is consistent but incomplete; therefore, the NL requirements need to be modified.

Modified NL requirements for Assembly System
Based on the analysis, the requirements are rewritten to resolve the identified incompleteness as below:

"An assembly unit consists of a user, a belt, a vision system, a robot, and a tray. The robot has two arms. The tray is used for assembly. The user, when ready, starts the assembly unit. The user places a dish and a cup on the moving belt. The belt conveys the dish and cup towards the vision system. The vision system has the capability to sense the entry of a part into the sensor zone. As soon as the vision system senses the entry of the dish and cup on the belt into the sensor zone, the vision system informs the belt to stop. The vision system then recognizes the type of part, i.e., whether a dish or a cup and informs the robot to pick it up from the belt using its arms. The robot then picks up the dish in one arm and the cup in another arm from the belt. The robot then places the dish in one arm and the cup in another arm on the tray. The vision system informs the belt to start moving, and the belt starts to move again. An assembly is said to be complete when a dish and cup are placed on the tray separately by the arms of the robot. The user is given the option to stop the assembly unit or to continue the operation of assembling the next tray with dish and cup".

Proof of Consistency and Completeness of the final PN model

The Assembly system PN (Fig. A.2.4) is now examined for consistency and completeness.

The property of consistency and completeness of the modified Assembly system PN is verified by running algorithm 1 and algorithm 2, respectively. By simulation of the PN, the execution of various transitions is found to be successful. The system is now consistent and complete.

Case study – Library system
Requirements Specifications

"Consider a small library database with the following transactions:
1) Check out a copy of a book. Return a copy of a book.

2) Add a copy of a book to the library. Remove a copy of a book from the library.

3) Get the list of books by a particular author or in a particular subject area.

4) Find out the list of books currently checked out by a particular borrower.

5) Find out what borrower last checked out a particular copy of a book."

"There are two types of users: staff users and ordinary borrowers. Transactions 1, 2, 4, and 5 are restricted to staff users, except that ordinary borrowers can perform transaction 4 to find out the list of books currently borrowed by themselves. The database must also satisfy the following constraints:

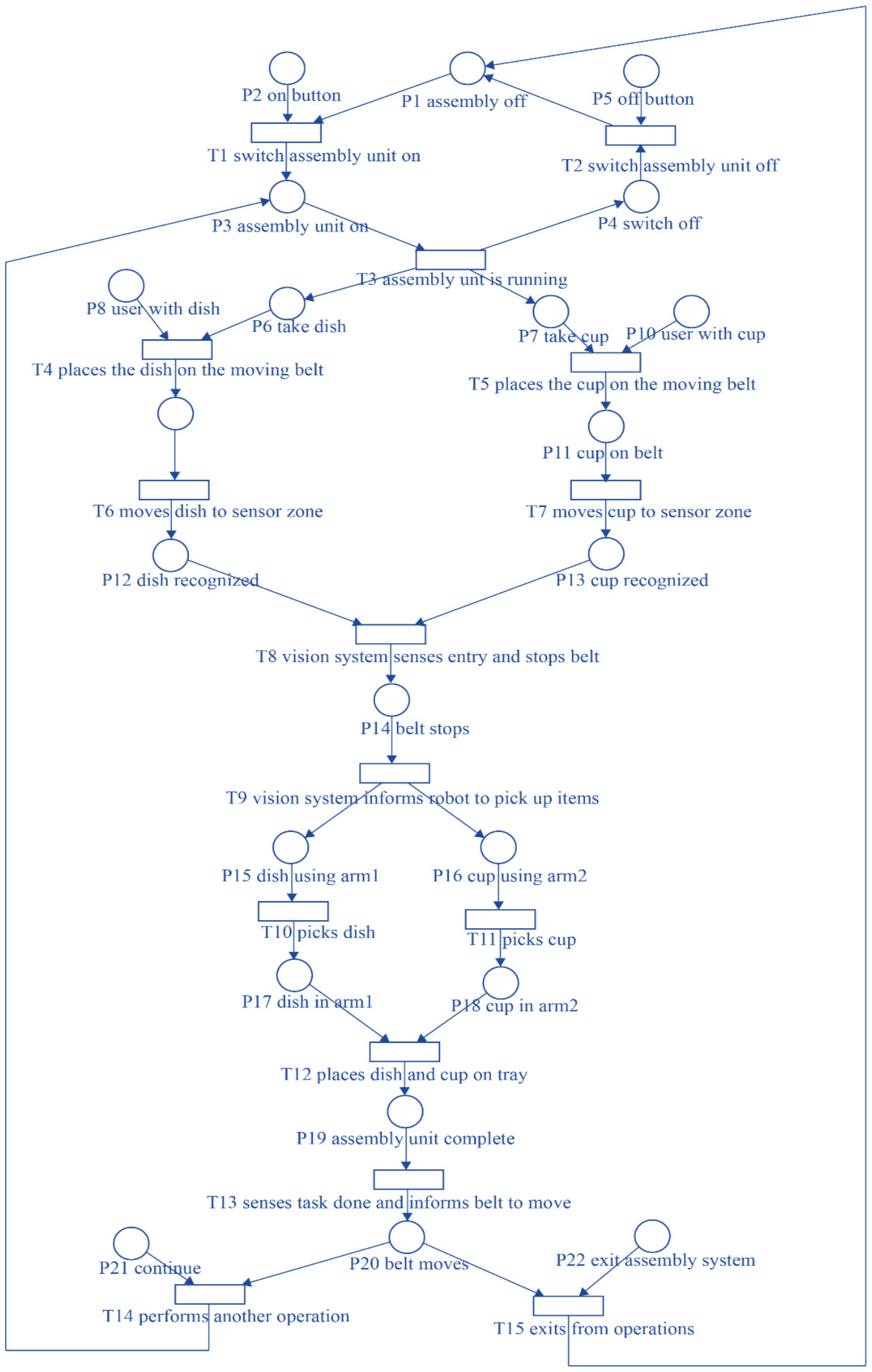
P2 on button
P1 assembly off
P5 off button
T1 switch assembly unit on
T2 switch assembly unit off
P3 assembly unit on
P4 switch off
T3 assembly unit is running
P8 user with dish
P6 take dish
P7 take cup
P10 user with cup
T4 places the dish on the moving belt
T5 places the cup on the moving belt
P11 cup on belt
T6 moves dish to sensor zone
T7 moves cup to sensor zone
P12 dish recognized
P13 cup recognized
T8 vision system senses entry and stops belt
P14 belt stops
T9 vision system informs robot to pick up items
P15 dish using arm1
P16 cup using arm2
T10 picks dish
T11 picks cup
P17 dish in arm1
P18 cup in arm2
T12 places dish and cup on tray
P19 assembly unit complete
T13 senses task done and informs belt to move
P21 continue
P20 belt moves
P22 exit assembly system
T14 performs another operation
T15 exits from operations

Fig. A.2.4 PN of Assembly system drawn after analysis and correction for consistency and completeness.

Five transitions - switch assembly unit on, switch assembly unit off, assembly unit is running, performs another operation and exits from operations have been added in Fig. A.2.4

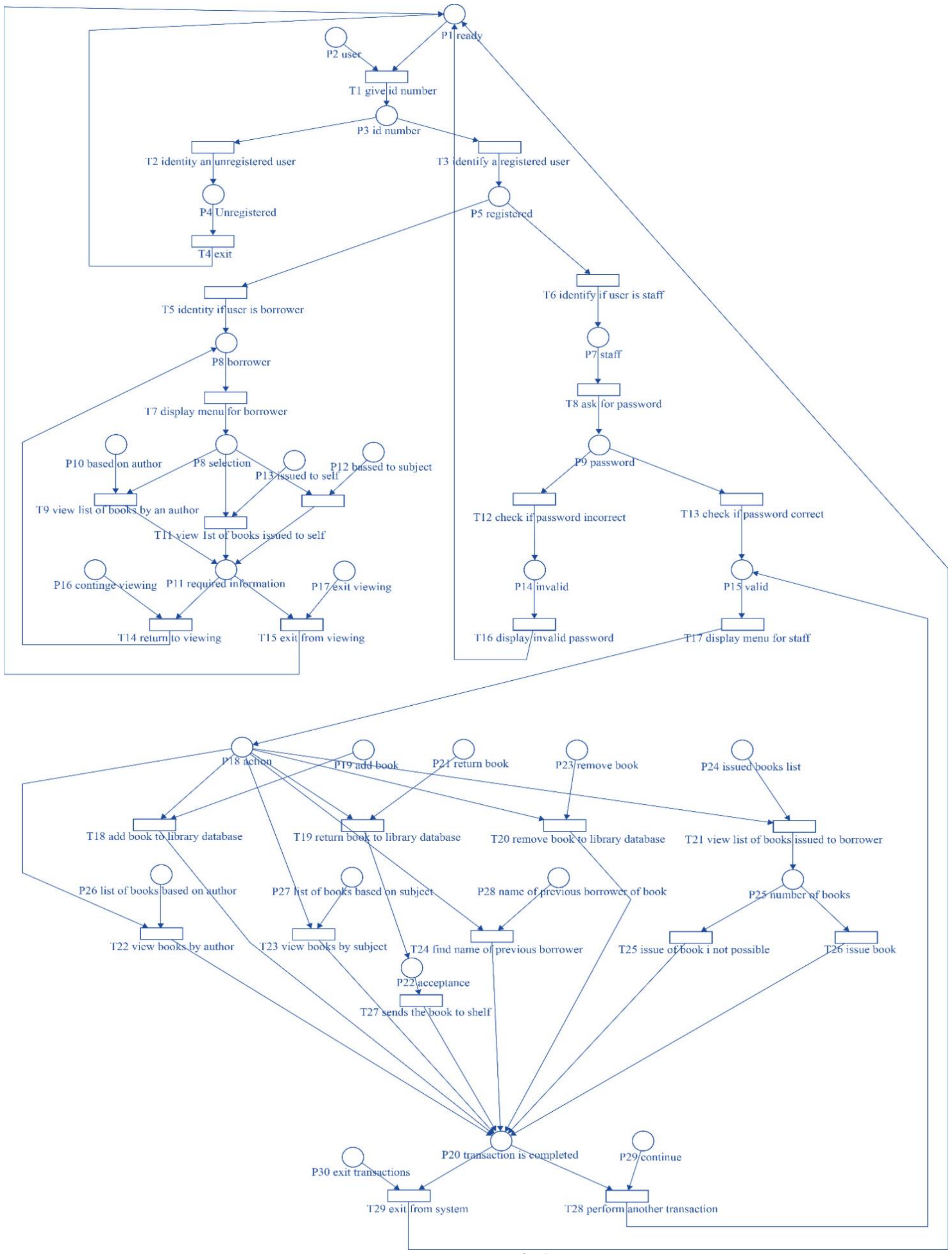
P1 ready
P2 user
T1 give id number
P3 id number
T2 identity an unregistered user
T3 identify a registered user
P4 Unregistered
P5 registered
T4 exit
T5 identity if user is borrower
T6 identify if user is staff
P8 borrower
P7 staff
T7 display menu for borrower
T8 ask for password
P10 based on author
P8 selection
P13 issued to self
P12 bassed to subject
P9 password
T9 view list of books by an author
T11 view 1st of books issued to self
T12 check if password incorrect
T13 check if password correct
P16 continue viewing
P11 required information
P17 exit viewing
P14 invalid
P15 valid
T14 return to viewing
T15 exit from viewing
T16 display invalid password
T17 display menu for staff
P18 action
P19 add book
P21 return book
P23 remove book
P24 issued books list
T18 add book to library database
T19 return book to library database
T20 remove book to library database
T21 view list of books issued to borrower
P26 list of books based on author
P27 list of books based on subject
P28 name of previous borrower of book
P25 number of books
T22 view books by author
T23 view books by subject
T24 find name of previous borrower
T25 issue of book i not possible
T26 issue book
P22 acceptance
T27 sends the book to shelf
P20 transaction is completed
P29 continue
P30 exit transactions
T29 exit from system
T28 perform another transaction

Fig. A.2.5. PN representation of Library System.

(1) All copies in the library must be available for checkout or be checked out.

(2) No copy of the book may be both available and checked out at the same time.

(3) A borrower may not have more than a prepare-defined per of books checked out at one time." [31]

Assumptions
The below-given assumptions have been made while developing PN as the required information has not been specified in the initial specifications:

(1) Both staff and borrowers need to be identified before they can use the library facilities.

(2) Database constraints (bullets 1 and 2) implementation will not be part of the PN execution.

(3) Database constraint 3 will be shown when checking is done before the issue of a copy of the book to the borrower.

(4) A borrower cannot be issued a book if he already has a copy of the same already issued.

First iteration - Analysis of PN drawn in Fig. A.2.5 using Algorithm 1 and Algorithm 2

Consistency analysis:

Work through Algorithm 1.

Step 1: List of places are:
1) ready
2) user
3) id number
4) unregistered
5) registered
6) borrower
7) staff
8) selection
9) password
10) based on the author
11) required information
12) based on subject
13) issued to self
14) invalid
15) valid
16) continue viewing
17) exit viewing
18) option
19) add the book
20) transaction is completed
21) return the book
22) acceptance
23) remove the book
24) issued books list
25) number of books
26) list of books based on the author
27) list of books based on subject
28) name of the previous borrower of the book
29) continue
30) exit transactions

Step 2: A place connected to more than one outgoing transition and where these transitions do not have any other input place connected to it or any information on their arcs is

1. "id number" (P_3) place and transitions <u>identify an unregistered user</u> (T_2) & <u>identify a registered user</u> (T_3) as seen in Fig. A.2.5.

The PN drawn in Fig. A.2.5 indicates inconsistent NL requirements.

Solution: From the application of algorithm 1 on the PN drawn in Fig. 1, it is seen that the two transitions viz. T_2 and T_3 are enabled together. Hence it is a case of inconsistency. Different priorities will be assigned for T2 and T3 to resolve this inconsistency problem as per the "Transitions with time" proposition. In this case, we assign T_2 a higher priority than T_3. This resolves the inconsistency in the PN. If the functional condition of T_2 is enabled, transition T_2 is fired; otherwise, after a duration $\Delta 1$, transition T_3 is fired.

2. "registered" (P_5) place and transitions <u>identify if a user is a borrower</u> (T_5) & <u>identify if the user is the staff</u> (T_6) as seen in Fig. A.2.5.

The PN drawn in Fig. A.2.5 indicates inconsistent NL requirements.

Solution: From the application of algorithm 1 on the PN drawn in Fig. 1, it is seen that the two transitions viz. T_5 and T_6 are enabled together. Hence it is a case of inconsistency. To resolve this inconsistency problem, different priorities are assigned for T_5 and T_6 as per the "Transitions with time" proposition. In this case, we assign T_5 a higher priority than T_6. This resolves the inconsistency in the PN. If the functional condition of T_5 is enabled, transition T_5 is fired; otherwise, after a duration $\Delta 2$, transition T_6 is fired.

3. "password" (P_8) place and transitions <u>check if the password is incorrect</u> (T_{12}) & <u>check if the password is correct</u> (T_{13}) as seen in Fig. A.2.5.

The PN drawn in Fig. A.2.5 indicates inconsistent NL requirements.

Solution: From the application of algorithm 1 on the PN drawn in Fig. A.2.5, it is seen that the two transitions, viz. check if the password is incorrect (T_{12}) and check if the password is correct (T_{13}), are enabled together. Hence it is a case of inconsistency. To resolve this inconsistency problem, different priorities are assigned for T_{12} and T_{13} as per the "Transitions with time" proposition. In this case, we assign T_{12} a higher priority than T_{13}. This resolves the inconsistency in the PN. If the functional condition of T_{12} is enabled, transition T_{12} is fired. Otherwise, after a duration, Δ3 transition T_{13} is fired.

4. "number of books" (P_{25}) place and transitions issue of the book is not possible (T_{25}) &issue book (T_{26}) as seen in Fig. A.2.51

The PN drawn in Fig. A.2.5 indicates inconsistent NL requirements.

Solution: From the application of algorithm 1 on the PN drawn in Fig. A.2.5, it is seen that the two transitions, viz., issue of the book is not possible (T_{25}) and issue book (T_{26}), are enabled together. Hence it is a case of inconsistency. To resolve this inconsistency problem, different priorities are assigned for T_{25} and T_{26} as per the "Transitions with time" proposition. In this case, we assign T_{25} a higher priority than T_{26}. This resolves the inconsistency in the PN. If the functional condition of T_{25} is enabled, transition T_{25} is fired; otherwise, after a duration of Δ4, transition T_{26} is fired.

Step 3: Simulate the PN to verify liveness. Simulation is successful, and hence PN is consistent with regard to liveness.

Step 4: Simulate the PN to verify that there are no dead transitions. Traverse all loops in the PN to determine transitions that are never enabled. There exist no transitions that are never enabled, and hence PN is consistent with regard to dead transitions.

The PN is inconsistent.

Completeness analysis:

Work through Algorithm 2.

Corrected list of Transitions

Step 1: List of transitions are:
1) give the ID number
2) identify an unregistered user
3) identify a registered user
4) exit
5) identify if the user is the borrower
6) identify if a user is a staff
7) display menu for the borrower
8) ask for a password
9) view a list of books by an author
10) view the list of books by subject
11) view the list of books issued to self
12) check if the password is incorrect
13) check if the password is correct
14) return to viewing
15) exit from viewing
16) display an invalid password
17) display menu for staff
18) add the book to the library database
19) return the book to the library database
20) remove the book from a library database
21) view the list of books issued to the borrower
22) view books by the author
23) view books by subject
24) find the name of the previous borrower
25) issue of the book is not possible
26) issue book
27) sends the book to the shelf
28) perform another transaction
29) exit from the system

The list of places are:
1) ready
2) user
3) id number
4) unregistered
5) registered
6) borrower
7) staff
8) selection
9) password
10) based on the author
11) required information
12) based on subject
13) issued to self
14) invalid
15) valid
16) continue viewing
17) exit viewing
18) option
19) add the book
20) transaction is completed
21) return the book
22) acceptance
23) remove the book
24) issued books list
25) number of books
26) list of books based on the author
27) list of books based on subject
28) name of the previous borrower of the book
29) continue
30) exit transactions

Step 2: Obtain a list of all functions.

1) switch on and switch off the library system
2) give the ID number to the user
3) identify a user as registered or unregistered
4) identify a user as a valid staff or borrower
5) display menu for staff
6) display menu for the borrower
7) ask and validate the password
8) return and exit to viewing
9) allow the borrower to view a list of books by an author, subject and issued to self
10) allow staff to add a book, return the book and remove the book from a library database
11) allow staff to view books by author and subject, find the name of the previous borrower, to view a list of books issued to the borrower (issue of the book is not possible when a borrower has crossed pre-defined limits or if the person has already borrowed a copy of the same book), issue book and send the book to the shelf
12) perform another transaction
13) exit from the system

Step 3: Traverse the PN and list functions absent in PN. One function, namely function 1 from step 2 above, is absent; hence, PN is incomplete.

Step 4: Obtained reachability report using PIPE2 to determine whether there is non-determinism in the PN. There exist no transitions that are not reachable. PN is complete with regard to non-determinism.

Step 5: Traverse PN. All places and transitions in the PN have distinct names. PN is complete with regard to distinct names.

Step 6: Traverse PN. There are no isolated subnets. PN is complete with regard to isolated subnets.

Step 7: Simulate PN. The PN executes in an infinite loop. This indicates that start and stop functionality is not defined. Since this is already determined by Step 3 above, we consider that the PN is complete with regard to infinite loops. The PN executes without abending.

Step 8: Simulate PN. There is no token accumulation. PN is complete with regard to token accumulation.

The PN is incomplete.

Modified NL requirements for Library System

Based on the analysis, the requirements are rewritten to resolve the identified inconsistencies and incompleteness as below:

"1) Check out a copy of a book. Return a copy of a book.

2) Add a copy of a book to the library. Remove a copy of a book from the library.

3) Get the list of books by a particular author or in a particular subject area.

4) Find out the list of books currently checked out by a particular borrower.

5) Find out what borrower last checked out a particular copy of a book.

6) If the library system is off, it is switched on before use. At the end, after using the library system is switched off."

"There are two types of users: staff users and ordinary borrowers. Transactions 1, 2, 4, 5 and 6 are restricted to staff users, except that ordinary borrowers can perform transaction 4 to find the list of books currently borrowed. The database must also satisfy the following constraints:

- All copies in the library must be available for checkout or be checked out.
- No copy of the book may be both available and checked out at the same time.
- A borrower may not have more than a prepare-defined per of books checked out at once."

7) Proof of Consistency and Completeness of the final PN model

The Library system PN (Fig. A.2.6) is now examined for consistency and completeness.

The property of consistency and completeness of the modified Library system PN is verified by running algorithm 1 and algorithm 2, respectively. By simulation of the PN, the execution of various transitions is found to be successful. The system is now consistent and complete.

Author contribution statement

AKB conceived, developed the methodology and performed formal analysis and investigation. AKB developed algorithms and propositions required. AKB wrote the manuscript and edited it after review. VKA and JR reviewed the methodology and supervision. All authors read and approved the manuscript.

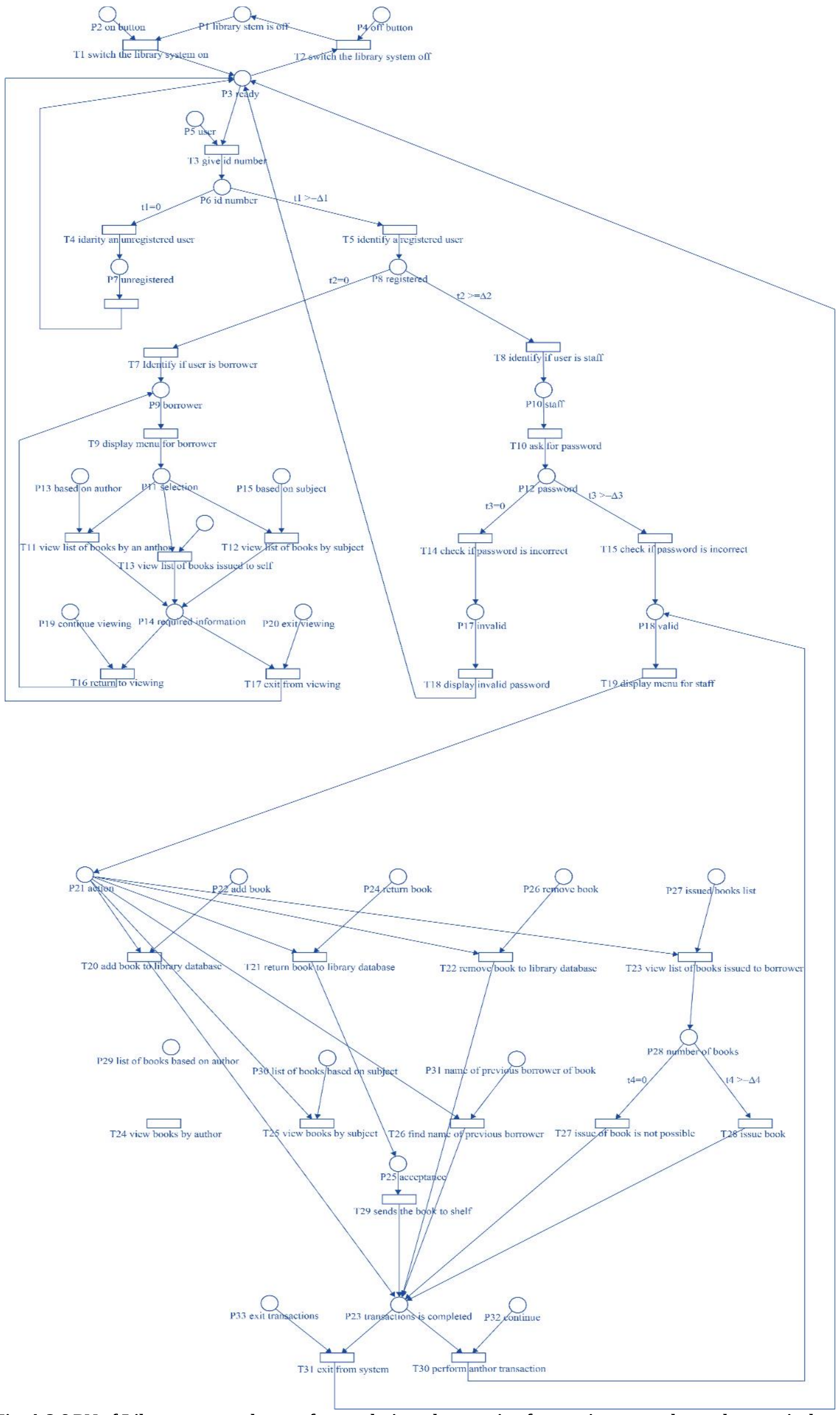
P2 on button
P1 library stem is off
P4 off button
T1 switch the library system on
T2 switch the library system off
P3 ready
P5 user
T3 give id number
P6 id number
t1=0
t1 >=Δ1
T4 idarity an unregistered user
T5 identify a registered user
P7 unregistered
P8 registered
t2=0
t2 >=Δ2
T7 Identify if user is borrower
T8 identify if user is staff
P9 borrower
P10 staff
T9 display menu for borrower
T10 ask for password
P13 based on author
P11 selection
P15 based on subject
P12 password
t3=0
t3 >=Δ3
T11 view list of books by an anthor
T12 view list of books by subject
T13 view list of books issued to self
T14 check if password is incorrect
T15 check if password is incorrect
P19 continue viewing
P14 required information
P20 exit viewing
P17 invalid
P18 valid
T16 return to viewing
T17 exit from viewing
T18 display invalid password
T19 display menu for staff
P21 action
P22 add book
P24 return book
P26 remove book
P27 issued books list
T20 add book to library database
T21 return book to library database
T22 remove book to library database
T23 view list of books issued to borrower
P29 list of books based on author
P30 list of books based on subject
P31 name of previous borrower of book
P28 number of books
t4=0
t4 >=Δ4
T24 view books by author
T25 view books by subject
T26 find name of previous borrower
T27 issue of book is not possible
T28 issue book
P25 acceptance
T29 sends the book to shelf
P33 exit transactions
P23 transactions is completed
P32 continue
T31 exit from system
T30 perform anthor transaction

Fig. A.2.6 PN of Library system drawn after analysis and correction for consistency and completeness is done.

Two transitions - switch the library system on and switch the library system off have been added in Fig. A.2.6

References

[1] Michel dos Santos Soares, and Daniel Souza Cioquetta, "Analysis of Techniques for Documenting User Requirements," *Computational Science and its Applications,* Berlin: Springer, pp. 16-28, 2012. *Crossref,* https://doi.org/10.1007/978-3-642-31128-4_2

[2] Michael Christel, and Kyo C. Kang, "*Issues in Requirements Elicitation*," Carnegie Mellon University, Technical Report, 1992.

[3] Robert Darimont, and Axel van Lamsweerde, "Formal Refinement Patterns for Goal-Driven Requirements Elaboration," *ACM SIGSOFT Software Engineering Notes,* vol. 21, no. 6, pp. 179-190, 1996. *Crossref,* https://doi.org/10.1145/250707.239131

[4] Shaoying Liu, "A Formal Structured Method for Requirement Specification Construction," *Proceedings of the 1992 ACM/SIGAPP Symposium on Applied Computing: Technological Challenges of the 1990's*, pp. 1055-1063, 1992. *Crossref,* https://doi.org/10.1145/130069.130130

[5] Shaoying Liu, "A User-Friendly Formal Requirements Specification Method," *Proceedings of the 30th Annual Southeast Regional Conference*, pp. 211–218, 1992. *Crossref,* https://doi.org/10.1145/503720.503802

[6] M. Osborne, and C. K. MacNish, "Processing Natural Language Software Requirement Specifications," *Proceedings of the Second International Conference on Requirements Engineering, Colorado Springs,* pp. 229-236, 1996. *Crossref,* https://doi.org/10.1109/ICRE.1996.491451

[7] Wenbin Li, "Toward Consistency Checking of Natural Language Temporal Requirements," *26th IEEE/ACM International Conference on Automated Software Engineering, IEEE,* pp. 651-655, 2011. *Crossref,* https://doi.org/10.1109/ASE.2011.6100148

[8] Claudio Menghi, "Verifying Incomplete and Evolving Specifications," *Proceedings of the 36th International Conference on Software Engineering, ACM,* pp. 670–673, 2014. *Crossref,* https://doi.org/10.1145/2591062.2591090

[9] Kirsten Sinclair, "*The Impact of Petri Nets on System-of-Systems Engineering*," Ph.D. Dissertation, Durham University, England, 2009.

[10] Jinqiang Zhao, and Zhenhua Duan, *Verification of Use Case with Petri Nets in Requirement Analysis,* Computational Science and Its Applications, Springer, pp. 29-42, 2009. *Crossref,* https://doi.org/10.1007/978-3-642-02457-3

[11] C. Vamsikrishna, and G. Padmanabhan, "Role of Petri Nets in Flexible Manufacturing System – A Review," *International Journal of Engineering Trends and Technology*, vol. 41, no. 2, pp. 90-100, 2016. *Crossref,* https://doi.org/10.14445/22315381/IJETT-V41P217

[12] Woo Jin Lee, Sung Deok Cha, and Yong Rae Kwon, "Integration and Analysis of Use Cases Using Modular Petri Nets in Requirements Engineering," *IEEE Transactions on Software Engineering,* vol. 24, no. 2, pp. 1115-1130, 1998. *Crossref,* https://doi.org/10.1109/32.738342

[13] K.S. Cheung, T.Y. Cheung, and K.O. Chow, "A Petri-Net-Based Synthesis Methodology for Use-Case-Driven System Design," *The Journal of Systems and Software*, vol. 79, no. 6, pp. 772–790, 2006. *Crossref,* https://doi.org/10.1016/j.jss.2005.06.018

[14] Stéphane S. Somé, "Formalization of Textual Use Cases Based on Petri Nets," *International Journal of Software Engineering and Knowledge Engineering*, vol. 20, no. 5, pp. 695–737, 2010. *Crossref,* https://doi.org/10.1142/S0218194010004931

[15] Edgar Sarmiento et al., "Analysis of Scenarios with Petri Net Models," *29th Brazilian Symposium on Software Engineering*, pp. 90-99, 2015. *Crossref,* https://doi.org/10.1109/SBES.2015.13

[16] Daniel Sinnig, Patrice Chalin, and Ferhat Khendek, "LTS Semantics for Use Case Models," *Proceedings of the 2009 ACM Symposium on Applied Computing*, pp. 365–370, 2009. *Crossref,* https://doi.org/10.1145/1529282.1529362

[17] Edgar Sarmiento-Calisaya et al., "Towards the Improvement of Natural Language Requirements Descriptions: The C&L Tool," *Proceedings of the 35th Annual ACM Symposium on Applied Computing,* pp. 1405–1413, 2020. *Crossref,* https://doi.org/10.1145/3341105.3374028

[18] R.Sunther, "Simplification of a Petri Net Controller in Industrial Systems," *SSRG International Journal of Industrial Engineering,* vol. 2, no. 1, pp. 4-7, 2015. *Crossref,* https://doi.org/10.14445/23499362/IJIE-V2I2P102

[19] W. M. P. van der Aalst, "The Application of Petri Nets to Workflow Management," *Journal of Circuits, System, and Computers*, vol. 8, no. 1, pp. 21–66, 1998.

[20] T. Murata, "Petri Nets: Properties, Analysis and Applications," *Proceedings of the IEEE,* vol. 77, no. 4, pp. 541-580, 1989. *Crossref,* https://doi.org/10.1109/5.24143

[21] Ian Sommerville, *Software Engineering,* 9th Edition, Boston: Addison-Wesley, 1995.

[22] Philip Meir Merlin, "*A Study of the Recoverability of Computing Systems*," Ph.D. Dissertation, University of California, Irvine, U.S.A, 1974.

[23] Thomas Hujsa, and Raymond Devillers, "On Dead Lock Ability, Liveness and Reversibility in Subclasses of Weighted Petri Nets," *Fundamentals of Informatics,* Polish Mathematical Society, vol. 161, no. 4, pp. 383-421, 2018. *Crossref,* https://doi.org/10.3233/FI-2018-1708

[24] B. W. Boehm, "Verifying and Validating Software Requirements and Design Specifications," *IEEE Software,* vol. 1, no. 1, pp. 75-88, 1984. *Crossref,* https://doi.org/10.1109/MS.1984.233702

[25] M.P.E. Heimdahl, and N.G. Leveson, "Completeness and Consistency in Hierarchical State-Based Requirements," *IEEE Transactions on Software Engineering,* vol. 22, no. 6, pp. 363-377, 1996. *Crossref,* https://doi.org/10.1109/32.508311

[26] Ngnassi Djami Aslain Brisco, Nzié Wolfgang, and Doka Yamigno Serge, "Maintenance Modularity Optimization using Clustering Algorithm: Application," *SSRG International Journal of Industrial Engineering*, vol. 7, no. 1, pp. 12-24, 2020.
Crossref, https://doi.org/10.14445/23499362/IJIE-V7I1P102

[27] Olaf Kummer et al., 2016. [Online]. Available: http://www.renew.de

[28] Connie U. Smith, and Lloyd G. Williams, "*Performance Engineering Evaluation of Object-Oriented Systems with SPE•EDTM*," *Computer Performance Evaluation Modelling Techniques and Tools*, Lecture Notes in Computer Science, Heidelberg: Springer, 1997.

[29] W. P. Stevens, G. F. Myers, and L. C. Constantine, "Structured Design," *IBM Systems Journal*, vol. 13, no.2, pp. 115-139, 1974.
Crossref, https://doi.org/10.1147/sj.132.0115

[30] Imran Sarwar Bajwa, and M. Asif Naeem, "On Specifying Requirements Using a Semantically Controlled Representation," *Natural Language Processing and Information Systems NLD*, Berlin: Springer, pp. 217-220, 2011.
Crossref, https://doi.org/10.1007/978-3-642-22327-3_23

[31] Jeannette M. Wing, "A Study of 12 Specifications of the Library Problem," *IEEE Software*, vol. 5, no. 4, pp. 66-76, 1988.
Crossref, https://doi.org/10.1109/52.17803

[32] Lian Yu et al., "Completeness and Consistency Analysis on Requirements of Distributed Event-Driven Systems," *2nd IFIP/IEEE International Symposium on Theoretical Aspects of Software Engineering*, pp. 241-244, 2008.
Crossref, https://doi.org/10.1109/TASE.2008.46

[33] A. K. Bharadwaj, V. K. Agrawal, and J. Reddy, "Transforming Natural Language Requirements into Petri Nets - A Semi-Automated Approach," *International Conference on Artificial Intelligence and Data Science,* 2022.