SEMINAR REPORT
Seminar:Embedded Systems and Robotics in Summer Term 2022

TECHNISCHE UNIVERSITÄT
KAISERSLAUTERN

Formal Verification with Petri Nets

Mohini Nandanwar
Technische Universität Kaiserslautern, Department of Electrical and Computer Engineering

Note: *This report is a compilation of publications related to some topic as a result of a student seminar. It does not claim to introduce original work and all sources should be properly cited.*

The formal verification of any system specifies if its behaviour satisfies a formal specification. The formal verification is widely used in the field of software programming, hardware designing, safety critical systems like avionics, automotive etc. Petri nets is one popular methodology which uses equivalence and model checking to prove correctness of system. In this report we will summarize Petri nets concepts, what are the analysis techniques and modelling approaches, some types of Petri nets and how failures can be avoided using this algorithms [10].

1 Introduction

Software industry has become a crucial part of human beings with great impact on our lives. Software engineering is revolving around all the business areas like Automotive, Defence, Avionics, Manufacturing, Medical field and the list goes on. With so much involvement in human life, it's of extreme importance that the software developed should be error free and has great performance, efficiency, accuracy meeting all the business and user requirements.If these software are deployed and planned improperly, the cost of fixing these mistakes soars. Therefore, having a system in place that allows for the early detection and correction of design flaws is really important.

Software are built using different programming languages, modelling approaches or rule based systems. It becomes necessary to use a formal approach for development. These formal modeling languages have a number of benefits, including minimizing ambiguity and vagueness in informal descriptions, enabling consistency and completeness checks, and bridging the line between informal models and system designs. Some claim that their practical usefulness is limited and that they lack the expressiveness needed for usage in practical applications. [5]

In this report we propose an approach based on Petri-Nets to verify the formal model. A Petri net is a formal language that enables the modelling of systems utilizing a solid mathematical foundation as a base. [5] [7] [9] The language is unique in that it makes it possible to model systems parallel, concurrent, asynchronous, and non-deterministic properties. For stepwise processes involving decision, repetition, and continuous execution, Petri nets offer a graphical notation. Petri nets offer a rigorous mathematical theory that enables process analysis using a variety of methodologies, as well as a clear mathematical explanation of their execution syntax. [12] [11]

The summary of this report is to explain Petri-nets as a formal verification method and how it can be used to formally verify different areas of software development. Along with this we will discuss about different types of Petri-nets like Timed and Coloured Petri-nets and its applications. Software development has following stages: Requirement gathering, analysis,

design, code, testing, deployment and maintenance. Early requirement analysis reduces excessive expenditures, complication, and complexity of the code in later stages of software development. [6]

The remains of the work is organized as follows: Section "Overview of the Method"gives an overview of the method Petri-nets, its different types used in related research papers and areas of application. Here we have considered for verifying rule based systems, verifying software scenarios and verification of use cases in requirement analysis phase of software. Furthermore in Section 3 we will have a quick overview of formal verification and different kinds of algorithms developed to avoid the faults or errors which may lead to failure of systems.Finally we will also look at the methods like TCPN model, reachability graphs, transforming scenarios into Petri-nets with sequencing diagram and more methods. Finally we will conclude with the advantages and disadvantages of verification and proposed future work using Petri-nets in Section 5 ..."

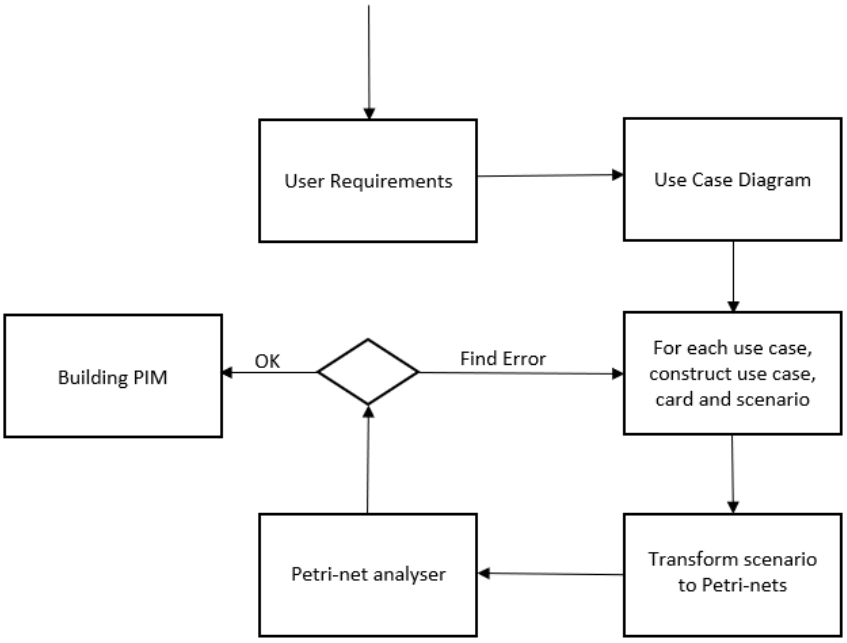

Figure 1: An overview of the approach [15]

2 Overview of the Method

2.1 Petri nets Fundamentals

Petri nets were created by Carl Adam Petri, who presented his work in his doctoral dissertation in 1962. It was beneficial and provided a number of ideas for updating the informatics

and computer science fundamentals.Petri wanted to show the effectiveness of asynchronous over synchronous systems. [12]This prototype led him to the conclusion that a general theory of processing of data must begin with asynchronous, locally confined processes if it is to be both practically applicable and not overly idealizing. Therefore, using sequential models as the foundation for informatics theory is incorrect. Apart from the modelling technique of asynchronous systems the Petri nets has further requirements justification. Petri advised taking full advantage of and respecting the location of activities during system runs by modelling them with the greatest degree of accuracy. [11] This characteristic is only implied when an action is described as a pair of an old state and a new state, as is typically done. To depict a single system run as a series of action events is simply unsuitable. [13]Petri advises that action events be arranged according to the partial order created by the relationship between cause and consequence rather than following a synthetic, unrealistic time scale. The order of the two action occurrences a and b may not change. If none of them are dependent on how the other one turns out, then this occurs. The absence of order then stands for concurrent. Be aware that a and b may be temporally independent of one another, with a preceding c in the sequence. Contrary to "situationally contemporaneous," concurrency is therefore not always a reflexive relation. Each is given a "pre-cone" and a "post-cone" according to relativity theory. [12] [11] These cones are made up of the events that occurred directly prior to and following a, respectively.

Petri advocates creating informatics models in the style of scientific models as a second prerequisite. Deep scientific theory is based on rules of preservation. Chemical compound equations and physics' principles governing energy conservation are two examples. A law of preservation should also apply to simple discrete actions. [11] The reactivity of processes is a typical essential requirement for conservation laws.According to Petri's third criterion, a modeling technique must not only be sufficient to represent effective implementation behavior but also people's practical utilisation of computing systems. It refers to both people's interactions with automata and the use of it to facilitate interpersonal communication. Petri listed these objectives and suggested ways to achieve them. [12] Theoretical informatics concentrated on computable foundations, formal languages, compiler theory, and sequential models for automaton and computers. Petri used technology to demonstrate the viability of his ideas, but his suggestions were too early for practical informatics. Over time, there was increased interest in decentralized and responsive systems, which helped Petri nets as well as a number of alternatives. [13] Partial order model checking was started by the concept that a decentralized run might be thought of as a partly ordered set of action occurrences. Traditional model checking is only able to handle entire orders, not part orders. However, a set of entire orders equates to a partial order. For all or none of the total orders in each collection, certain temporal logic properties are true.Petri Nets are a well-known modeling and analysis method for computer embedded systems. [12] Petri Nets are used by a sizable community of scientists and software engineers in a variety of applications.

Petri nets are a formalized language for designing and evaluating parallel and distributed systems that is frequently employed in a variety of academic disciplines. Petri nets can take on many different forms, from low-level Petri nets for building control systems to high-level Petri nets for building system data to tiered and object-oriented Petri nets for building pore structures. [13] A Petri net has places, transitions, and arcs. [1] Arcs move from one thing to another;

they never move between places or between changes in state. Input locations, as contrast to output places, are established when an arc moves toward a transition rather than away from one. A basic nets course of nets is expanded by the use of Petri nets, which are state transition systems. [6]

Formal Definition and basic Terminology of Petri nets: [1]
A place/transition net is a tuple (S, T, F, M_0, W, K)
S is set of places
T is set of Transitions
F is the flow relation
M_0 is initial marking
W maps arcs to positive numbers (arc weights)
K maps places to positive numbers (capacity restrictions)
The aforementioned items must adhere to several limitations, including $M_0(s) \leq K(s)$ for each place s [1]

Definition 1. A net is a tuple $N = (P, T, F)$ where
1. P and T are distinct bounded sets of places transitions.
2. $F \subseteq (SXT) \cup (TXS)$ [1]

In coming sections we discuss types of Petri nets as follow:

2.2 ω-Petri nets (ωPN)

ω-Petri nets (ωPN) are an extension of plain Petri nets PN in which instead of natural number input and output arcs are labelled by an ω symbol. [2]While ω output arc generates quasi in its output location, an ω labeled input arc intakes any number of tokens non-deterministically in its input place. [7]

- Syntax of $\omega-$ nets:
  An $\omega-$net N is a tuple (P, T, F, W, M_0) where
  1. (P,T,F) is the Petri net model,
     a) P and T are non-empty finite sets justifying $P \cap T = \varnothing$ (P and T are set of places and transitions respectively),
     b) $F \subseteq (PXT) \cup (TXP)$ is a relative (the arcs of N); [7]
  2. W is a mapping from $F \rightarrow 1$, each arc is assigned a flow capacity of 1. We don't explicitly display 1 on labels by default in ω-Petri nets
  3. $M_0 : P \rightarrow 0, \omega$ is a start, which has assigned value of 0 (no token) or ω) (any number of tokens) to predicate p in P. ω for any integer n, and furthermore $\omega + \omega = \omega$ and $\omega - \omega = \omega$ [2]
- Dynamic semantics of $\omega-$ nets

1. Markings of an $\omega-$ net are mappings $M : P \to \{0, \omega\}$ [9]
2. An $\omega-$ net's sequence $M0T0M1T1..$ is either infinite or finite, with each Ti being an execution step made up of a group of concurrent firing transitions when the last marking is terminal (no more enabled transitions in the last marking);
3. The list of all potential execution paths beginning with the first marking is how a $\omega-$net behaves. [7]

2.3 Coloured Petri Nets

A graphical language for system design, specification, simulation, and verification are called Coloured Petri Nets (CP-nets or CPN). In Figure 2 a diagram to explain CP-net is mentioned. It provides a straightforward transport protocol for sending and receiving a number of packets over an unstable network. Places refers to the ellipses and circles. They explain the system's states. Transitions are the rectangles. [5]The actions are described. Arcs are the name for the arrows. The transitions' effects on the predicament of CP-net are described by arc expressions.

There are a number of markers, or "tokens," in each place. Each of these tokens carries a data point as opposed to low-level Petri nets. Value that is a member of the specified type. Use Send (on the upper left) as an example.Fig. 2 bottom left corner shows seven tokens in their initial state and token value belong to the INTxDATA type represent's ready packets being sent. [8]

In contrary to the tokens of low-level Petri nets, which are traditionally shown as dark, "uncolored" dots, colored Petri nets are named in such a way that they may be distinguished from one another due to the use of tokens that contain data value. [8]

In the beginning, seemingly small, unstructured groupings of colors were used, such as counting a predetermined number of forms. After that, it was realized that the theory and the tools could be combined in such a manner that arbitrary sophisticated information types could be used as color sets. [5] These days, it is not unusual to even have tokens which hold sophisticated information, such as a list of thousands of entries with a wide variety of fields. A token color and it's value are no longer distinguishable from one another, and neither are a color set as well as a sort. [8] [5]

A transition must have sufficient tokens on its input locations for it to take effect. Let's take the example of SendPacket. It is surrounded by three arcs, two of which are double arcs. The variables n and p in the Three Arc expression are both of type $DATA$. In order for a transition to occur, these two variables must be constrained to values that correspond to their types in a way that each incoming arc's expression compares to a token value that is present on the input data. [8] As there was only one token with Value 1 in $NextSend$, it was implied that it would be linked to 1. Next, we observe that P should be connected to "Modellin," as Send only possesses one token, the first element of which is 1. This indicates that by adding the first Packet to the organize's input buffer, we have already dispatched it. We don't raise the NextSend counter or remove the packet from Send. The packet could be displaced on the arrangement, which is the cause. Therefore, we might need to resend the packet. Our method is pessimistic in that it continuously transmits the same packet until it receives a confirmation for the next bundle.

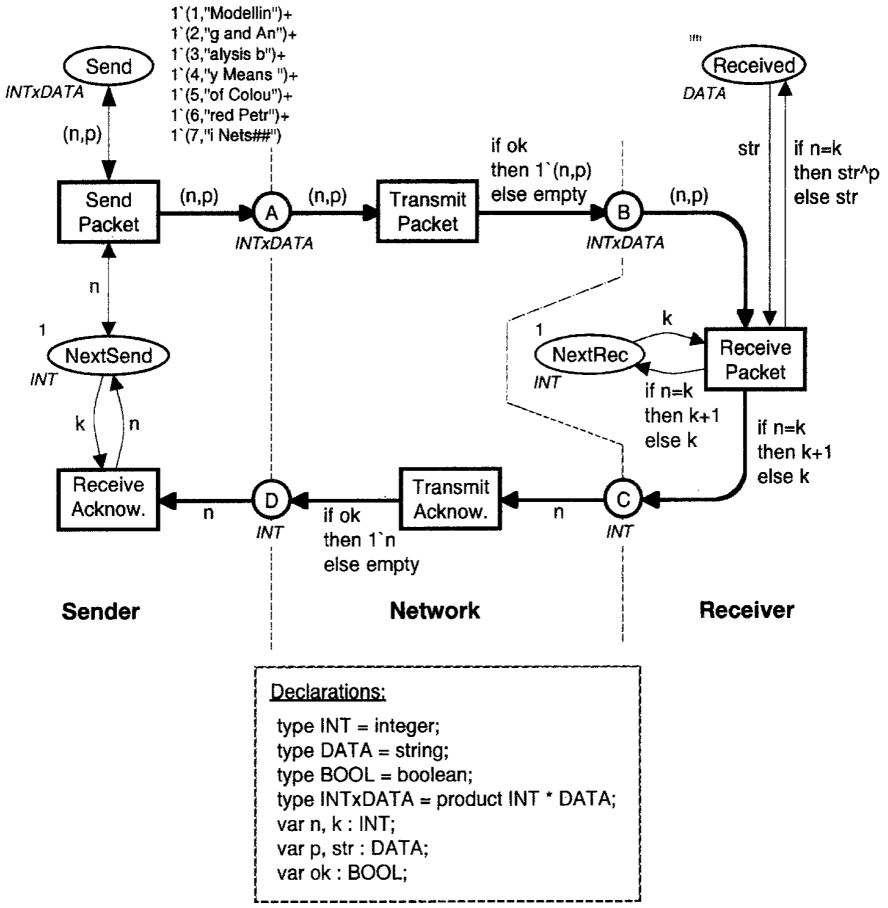

Figure 2: Basics of Coloured Petri Net [8]

Transitions *ReceivePacket* become active when a token reaches Place *B*. It matches the number k in the *NextRec* counter with the number n in the upcoming Packet. In the unlikely event when the two values are identical, the bundle is the one that was expected. The *NextRec* counter is then increased by one, an assertion is transmitted via Place *C*, and the data substance of the bundle is added to the previous data substance *str* of the token in place *Gotten*.If there is a difference between the two values, the packet is ignored. A confirmation is transmitted via Place *C*, and the values on *Gotten* and *NextRec* remain the same. Transition Similar to how *TransmitPacket* operates, *TransmitAcknow* transmits data. Affirmations are moved from location *C* to *D* and transmitted over the arrangement unless ok=false, where in affirmation is lost. *ReceiveAcknow* transitions handle lost acknowledgements. [8]

- **Analysis of Coloured Petri Nets:**

Re enactments are used to validate the CPN model, or to verify that it behaves as expected, while a CP-net is being developed.Early on, the CPN demonstration is simple, addressing

only a few aspects of framework and neglecting numerous different points of view regarding the final system. This suggests that plan errors can be caught early on and that the designers can gather underutilized information about the framework, information that can be applied to the remaining portions of the plan process.It is possible to use scripted reenactments and to highlight time delays when a CPN presentation has been debugged. They can be extremely rapid, with a few hundreds or thousands of transitions taking place every second, and are comparable to program executions in speed. The length of the various actions within the illustrated framework is also described. As an example, we could investigate how a convention's retransmission delay affects throughput and the efficient use of transmission capacity, or we could assess the effectiveness of two different conventions. [8]

The simplest form of verification relies on demonstrate checking through the use of state spaces, which frequently grow to enormous sizes. Therefore, having effective tools for building and inspecting state spaces is essential. [8] [5] Another confirmation method is based on location invariants, which are similar to the invariants used in program confirmation in many ways. In terms of the needed skills of the included labor force, this technique is more demanding, and as a result, invariants are seldom ever used in industrial endeavors.

2.4 Timed and Controlled Petri-net (TCPN)

A TCPN adds time to Controlled Petri-net. TCPN=(P, C, T, F,M_0, λ, μ, γ) where [9]

- P=$\{p_1, ...p_n\}$ is a finite set of state places
- C=$\{c_1, ...c_n\}$ is a finite set of controlled places
- T=$\{t_1, ...t_n\}$ is a finite set of transitions
- $F \subseteq (PXT) \cup (TXC) \cup (TXP)$ is relative (the arcs of N);
- $M_0 : P \rightarrow (0, 1, 2....)$ is an initial marking
- λ: $P \rightarrow N$ is a token function of state places
- μ: $C \rightarrow (0, 1)$ is a token function of control places
- γ: $T \rightarrow R^+$ is a time delay function [9]

3 Related Work

3.1 Errors in rule-based system's structural design

A rule-based system relies on knowledge obtained from subject matter experts. They are written as if - then conditional statements, thus checking as rule applier and pattern matching and written as inference rules. The most commonly encountered errors in rule-based system's structural design are inconsistency, redundancy, incompleteness, unreachability, subsumptions and circularity. [7]

Inconsistency: This procedure can be used to infer both the fact p and its negation q, which can result in contradictory facts.Following is a syntax for inconsistency method: [7]

$$R_1 : p \rightarrow q$$

$$R_2 : p \rightarrow \neg q$$

Incompleteness: It happens when the goal is not used as condition in any other rule. This may happen when some goals are unreachable or the result is dead-end. As it indicates that some important rules are missing, therefore useful information cannot be inferred from the methods. Lets say p is must for satisfying condition, [7] there are not rules like this: $\rightarrow p$

Redundancy: Whenever there are duplicate or merged rules in the method, it happens. It also generates interference and unnecessarily expands the method. Example is: [7]

$$R_1 : p \rightarrow q$$

$$R_2 : p \rightarrow q \wedge r$$

R_2 dominates R_1

Circularity: It happens when numerous rules have circular dependencies. It should be broken as it causes infinite reasoning. Example is: [7]

$$R_1 : p \rightarrow q$$

$$R_1 : q \rightarrow p$$

Self-circular is a exceptional case: $p \rightarrow p$

3.2 Visual Verification methods

There are n number of graphical techniques already existing which are used for formal verification of rule base systems. Some of them are directed graphs, hyper graphs, directed hypergraph etc. These are employed to identify the structural flaws listed in Section 3.1. In all of the aforementioned techniques, clauses are represented by nodes, and causal links are represented by directed edges. The disadvantages of these methods are some of them are not able to identify the structural errors properly and report errors. The Petri-nets can detect structural errors by comparing to predefined syntactic model. Directed hyper graphs uses adjacency matrix technique, but the drawback is it is costly in computation both in terms of time and space. Also this verification approach has many rows and columns for several operations which is also not easy to understand. [7] [2]

To research rule-based systems, many forms of Petri-nets have been utilized. They are used to detect conflicting rules, which could identify the inconsistency from multiple databases. Its main goal is not to identify all the rules mention in Section 3.1. In this report we will discuss about how Petri-nets is use to verify rule based systems. Each rule base is represented as a Petri-net, where inputs are transitions and antecedents, and outputs are the corresponding outcomes. An additional arc was placed between t and p in order to guarantee the veracity of an antecedent p of a transition t expressing an original rule. An additional input location with a

start token was given to the rule so that every subsequent rule may fire, which ensures that a rule can be run exactly once. A series of transitions which are mutually exclusive rules were added with an output place to designating the inconsistent state in order to discover consistency. [2] Conflicting rules are discovered when multiple tokens are present in an inconsistent location. But this method also has some limitations, it fails to identify which transition sequences results in firing and which transitions are not firable.Calculating the accessibility of submarkings was suggested using a linear equations analysis based on a transition matrix. Though there are drawbacks of this technique, it was still used. The restrictions include the failure to recognize the transition patterns that caused the labels and erroneous solutions that use transitions that aren't firable. The work proved a number of claims regarding the Petri net representation's capacity for error detection, but it did not offer a method for doing so. [7]

3.3 Requirement Scenarios of Software Applications

A scenarios can be defined as a description of the actions and reactions humans have when attempting to use computer systems and applications. [9] As scenarios are a crucial part of requirements engineering, it becomes of uttermost importance to verify them and to rectify if any ambiguous or missing data occurs in those scenarios. We will be summarizing how Timed Petri-nets or TPNs can be used to act as a method for their verification. It has been noted that scenarios have a number of benefits, such as expressing the application's user perspective, its easy to understand and communication with the users. It plays important role in software development lifecycle specially in requirements and analysis phase like support in architectural designing and requirements elicitation. [14] Different features of scenarios can be listed as below: [9]

- Representing view of a customer who is unaware of software's internal structure.
- Describing part of an application and not its whole behaviour.
- Under time constraints include explicit time intervals.
- Representing a sequence of user actions with different user roles and access as a whole and not as an individual situation. [9]

In our approach we will be using sequence diagrams in UML to represent them. Figure 4 depicts a typical sequence diagram where the vertical line serves as the object's lifeline and the rectangle serves as the object. The horizontal rows are the information or messages carried by the events. The solid arrow represents the event which happened inside an application, the synchronization factor is not taken into consideration. Messages can be classified as synchronous or asynchronous broadcast. The full arrowhead is a synchronous broadcast message. In this the transmitter waits until the message completes processing. The return message can be skipped and it has a pair after every subordinate message. The half arrowhead is an asynchronous message, which the sender is not prohibited, and keep sending. Also it should have an explicit return message. [9]Timing markings are used to specify the timepoints at which events or messages occurred, and relational expressions can be used to specify the temporal constraints within these timing marks.

In order to see the correctness of a scenario two types of errors are taken into consideration:

1. Missing Information: A scenario should be given along with all the used events and objects (both source and target objects). Three instances of missing data have been found:
   - There is no mention of the event's or object's name.
   - Not provided is the source or destination object.
   - Nothing enters or exits an object. [9]
2. Wrong Information: Scenarios that, either of conflicting circumstances or erroneous temporal limits, are described in an unattainable state. There are three instances of inaccurate information:
   - Timing restrictions that differ between two scenarios or within a single scenario
   - a lack of certainty within scenarios
   - improperly defined timing restrictions [9]

The scenarios are validated using time Petri-nets. The intended verification model for that is the TPNs model. TPN can be used to analyse modern systems whose behaviours are dependent on explicit periodic parameters. Because TPNs are a development of the conventional Petri networks paradigm, they may handle systems that include or lack periodic aspects. To formally detect and analyse the problems identified in scenarios, TPNs offers a broad analysis capability. [14]A transition in time TPN is connected with a set of parameters called delay and time-out, where the first parameter specifies how long the transition must wait before firing and the second value specifies how soon it must start firing. It means that if a transition is allowed at a specific time, it must occur between time plus delay time and time plus time out time, without firing earlier or later than that. A transition must fire at the timed-out time if it does not fire during the interval. [9]

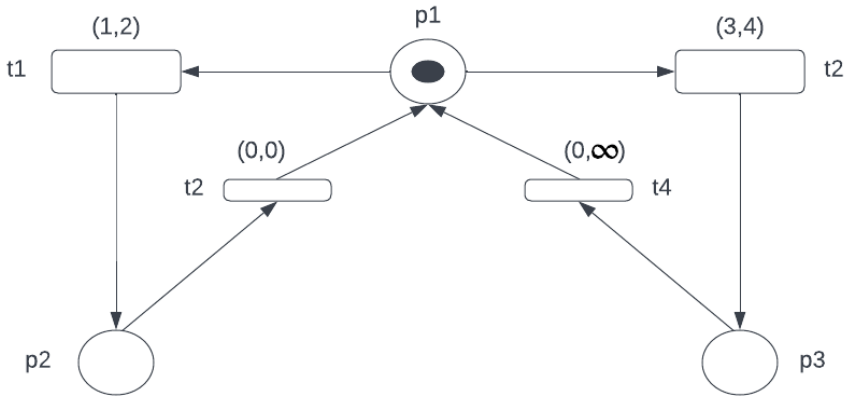

Figure 3: A diagram showing the timed Petri nets [9]

Refering the diagram shown in Figure 3, we can see how the TPN models work. [9] The transitions *t1* and *t2* are linked to a pair of (1, 2) and a pair of respectively (3,4). As a result, once *t1* is activated, it is required to shoot by five units of time and cannot fire before two units

of time. If a transition is connected to a pair of (0,0), like *t2*, it will start firing right away. (0, ∞) timing pairs, such as *t4*, are the standard. When all temporal pairs of transmission are (0,∞), the traditional petri-nets are a classical instance of TPNs

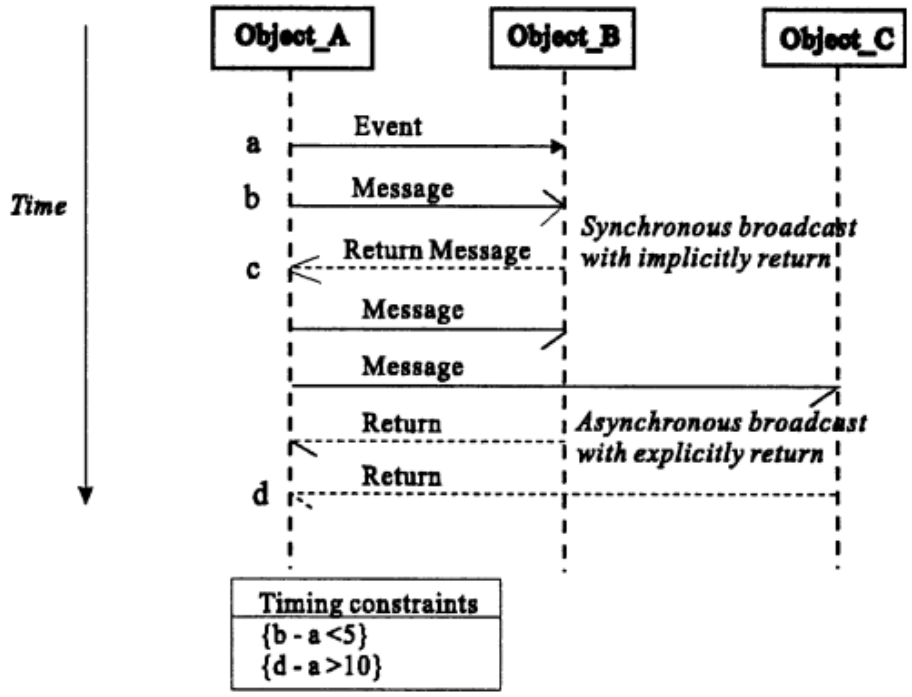

Figure 4: An example of sequence diagram [8]

3.4 Model Driven Petri-net Methodology

The different phases in SDLC are Requirements gathering, Analysis, Design, Coding, Testing and deployment. Many researchers have shown the considerable influence of earl detection of defects and faults in requirement and analysis phases. It helps in unnecessary costs, confusion, complexity and potential risks in the later SDLC stages. The use case approach, which is built on the idea of scenarios, is frequently employed as a method for requirement eliciting. Use cases are used to describe the software requirements. Each of them is written from the viewpoint of the user and is explained by a particular sequence of system events. [6]During the software development process, the use case technique has various useful benefits. On the one part, use case diagrams allow software users to determine whether the system meets their requirements right away during the software development process. They can contact requirements developers directly with their system-related problems, who can then make the appropriate adjustments to the requirements model. Before writing any code, users can assess the nature of system using use case diagrams. However, use case diagrams can serve as a guide throughout the entire software

development process.

However, some faults are still there in this use case approach.First is it is written in informal language and second it is difficult to analyse the dynamic behaviour to capture consistency and concurrency. Here we will use CIM to elicit the requirements. [6] [1] Petri nets enable the modeling of a system's resource distribution, synchronization, and parallelism behaviours. It Petri net has a representation that is visual and simple to comprehend also a clear syntax and a strong mathematical base. A variety of sophisticated analytic tools, including accessibility, conflict, restricted, safe etc., can be offered by a petri net. There are a few software applications that can help with Petri net analysis and simulation. Petri nets with UML connection may make it possible to effectively develop system agnostic models and automate evaluation in Model Driven Development. [1] [3]

The model-driven approach offers the chance to create a methodology-supporting framework. On the other side, Petri Nets help with service validation and act as a formal specification of the service functionality, or the service activities. [1]

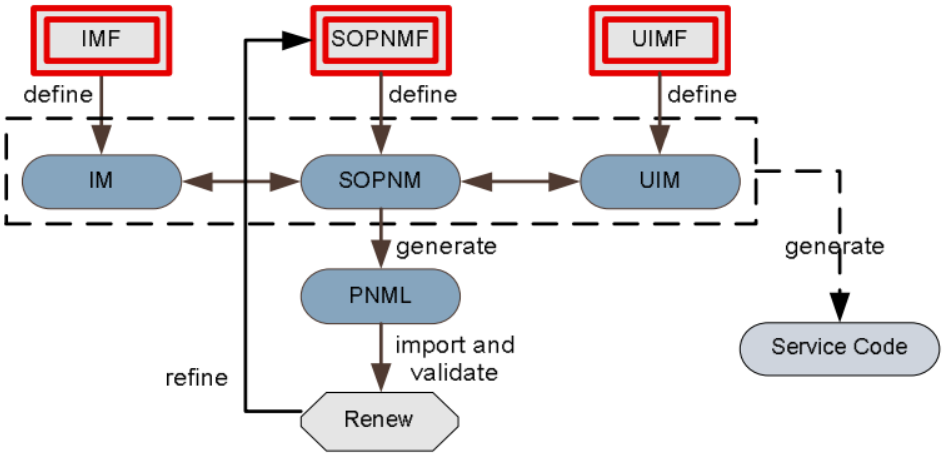

Figure 5: Model driven Petri-nets Methodology [1]

In the context of MDD, models are recorded using languages that are often used by the software industry in machine-readable representations. As a result, it enables them to interface with and reuse such models with various entities. Shorter development cycles and cheaper software production costs follow from this. Here, MDD is develops a technique that takes advantage of Petri Net and Sequence Diagram advantages. [1] [3]The method for service analysis, design, validation, and code creation is shown in Figure 3. The primary definition of the basic MDA meta model for the SOPN modelling language. It makes it easier to define service behaviour in the form of a PN that can later be verified. [3]

In this report, we will take into account use case cards, which are used to express the attributes of each use case, such as its name, actors, pre and post conditions, and some of its event flow. Each scenario can be represented as a TCPN by these, which are integrated into the scenarios. The procedure will restart with the reconstruction of the use case scenarios

and descriptions if any flaws or errors are found after these Petri net models have been verified; otherwise, models at the level of PIM will be developed by looking at these models in requirement analysis. [3] Figure 1 depicts this strategy in broad strokes.Petri nets are the technology we employ due to its mathematical simplicity, modelling flexibility, localization including both states and actions, and visual presentation, in addition to its well-created qualitative and quantitative analysis methodologies, and the accessibility of advanced analytic.

The technique is used in MDA, and the work is based on the MDD perspective. The use case card contains the criteria that we verified, and the event flow is represented in the revised scenario language. To make the Petri net model sufficiently regulated to depict the chosen behaviour and message exchange between objects, timed and controlled Petri nets are utilized. By examining the Petri net models, the parameters for demand quality assessment are created and the requirement model is validated. [15]All methods, however, fall short of perfection. The Timed and Controlled Petri Nets will encounter difficulties when the use case's complicated event flow is applied to industrial systems. If so, the use case presented via TCPN will not only be challenging to comprehend but also vulnerable to changes. Thankfully, the "including" relationship provides a strong hint that some event flow components in the use case can be separated and combined to create a new use case that has a "include" relationship to the original use case. Maybe we can link it to the OOPS concept like Inheritance. [15] [1]

3.5 Requirement Elicitation

Researching and learning the requirements of a system from users, customers, and other stakeholders is known as requirements elicitation. The procedure is also known as "requirement gathering" on occasion. Any new application's development needs to include requirement elicitation. The majority of the system fails as a result of poor elicitation decisions. Without the use of an elicitation approach, requirements and demands of users cannot be ascertained. Ensuring efficient communication between analysts and users during the elicitation process is the most frequent difficulty for analysts. The majority of errors occur in the system as a result of inadequate user-analyst communication. [15]The phrase "requirements" refers to the notion of requirement elicitation, which is described as the initial input statements of required functionalities and also the technique of expression of a user to articulate its problem. The primary requirements should be the focus of the requirement elicitation prototype, which is to be given to the customers.

By applying the purpose of elicitation requirement technique, it may be simpler for stakeholders to obtain the best and most appropriate application in accordance with all feasible needs. Due to the fact that it determines and specifies what needs to be built, Requirements engineering (RE) is one of the most challenging phases of the software development process. Communication with stakeholders, negotiating, and technique selection are all essential steps in the requirements elicitation process in order to ascertain the needs and desires of the various stakeholders. [15]This process is closely tied to the environment in which it is conducted, the particulars of the project, the surroundings, the adviser's expertise and knowledge. [4]The primary goal of the elicitation technique is to identify as many difficulties as possible. Direct and indirect methods are the two main techniques used to elicit the requirements. Interviewing is centerred on questioning the subject-matter expert about the area of interest as well as how

they carry out their duties. Interviews can take on any shape, including the formats used to distinguish between unstructured, semi-structured, and structured interviews. [4] Prototyping is an alluring and dynamic procedure that plays a significant role in the analysis stage of the SDLC in the actual world. Because prototyping can change the most fundamental things, it can prolong the information gathering process. With their assistance, we may get user input. Users can view the facilities and respond, and system analysts can then assess the comments and alter the current needs as well as create new ones. The idea of planning concept is improved by developers through prototyping and analysis of how it will function in real life.Some indirect types of elicitation techniques could be questionnaire where a set of pre defined questions are passed on to a group of expertise people who give their feedback based on historical experiences. Laddering is another technique in which a tree type structure is created which could show different stages of the RE like create, review and update. When already existing problems are encountered for the processes, ethnography method could be used. [4]

Here we will explain how to build a requirements model using TCPN and MDD development. An email client system is used to explain this approach.

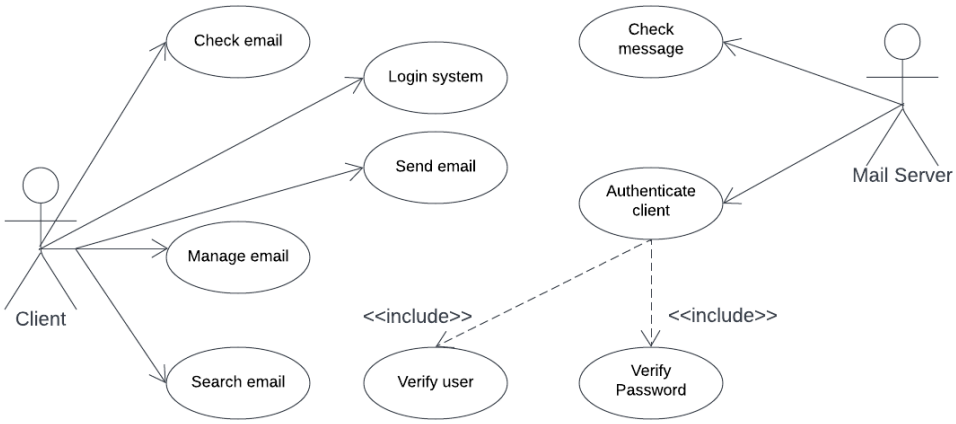

Figure 6: Use case illustration for a mail client system [15]

Figure 6 shows the use case diagram of email client services. A scenario and use case card are used to explain each use case. The scenario languages has advantages like it is easily understandable and syntactic rules allows transformation from description to TCPN feasible. The syntactic rules can help detect errors.Here are some syntax guidelines that will be used when creating scenarios. [15]

1. Every sentence has only one subject and predicate and no conjunctive words. [15]
   - Client sends ‘Packet’and ‘Terminate’to mail server; (False)
   - Client sends ‘Packet’;(True)
   - Client sends ‘Terminate’;(True)
2. Always use active voice sentences. [15]

- ‘Packet’is mailed to mail server;(False)
- Client sends ‘Packet’to mail server; (True)

3. Don’t duplicate keywords used for description, keep consistency in using descriptions. [15]
   - Client sends ‘Packet’to mail server; (True)
   - Customer sends ‘Terminate’to mail server; (False)
   - Client sends ‘Terminate’to mail server; (True) ‘Client ’and ‘Customer’are description of same thing.
4. Don’t use pronoun for Objects, instead use the Name. [15]
   - Client sends ‘Packet’to it; (False)
   - Client sends ‘Packet’to mail server;(True)

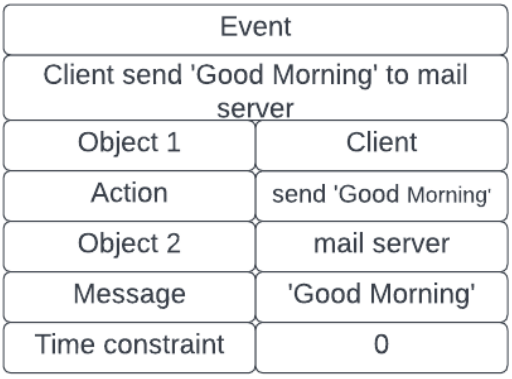

Figure 7: Event’s Frame [15]

5. Each event has an individual event.Figure 7 [15]
6. Each event terminates with a semicolon.
7. Sequence, selection, iteration, and concurrency are the relationships between individual events.
8. We employ "IF-THEN-ELSE {Condition, F1, F2};" when dealing with selective events. If the condition returns true, we move to F1, else we move to F2. The command to begin the condition check is modeled by the firing of transition.It enables the start of checking and places a token into a position. A token is automatically deposited into control spot after the check is finished. Whether the considered condition was true or false depends on the outcome. [15]
9. We use "DO-WHILE {Condition, F1, F2...};" for iterative events, doing F1, F2 repeatedly until "Condition" returns false.A token is placed in position as soon as a transition is fired, which also initiates the processing of event flow F1. The start of the condition check is modelled by the firing of transition . It places a token in position A token will automatically be inserted into control spot at the conclusion of the check. [15]

10. We use "And{ F1, F2, F3...};" for concurrency events.A token is inserted into both places by the firing of transition. Both the e vents must be carried out simultaneously in this manner. Only when tokens are present at both the places may transition be fired. The concurrent event is over when a token is presented inside location which is the target. [15]

4 Validation and Solution

4.1 Verification of rule base system using reachability graph

A reachable marker of the connected ω-net is represented by a vector, where each element denotes the status of a certain location. The set of accessible markers indicated by the aforementioned vectors form the nodes of a reachability graph, whereas the edges are unique sets of simultaneously firable transitions. [7] [2]

- Steps are: [7]
  1. Create initial node by setting value of V to ωs as inputs and 0s for the remaining.
  2. If no RT transition is enabled, stop, else let $\{t_1....t_k\}$ be list of all transitions that are enabled using the M by V, and label Ł$_1$ $= \varnothing$ and Ł$_2$ $= \varnothing$
     2.1 $i = 1....$ k, enablement of transitions produces marking M^l, then V is updated wrt M^l, sum t_i with L_1; else sum t_i with L_2;
     2.2 If Ł$_1$ $\neq \varnothing$ Create a node using V, and connect the previous node with an edge marked L_1;
     2.3 If Ł$_2$ $\neq \varnothing$ connect edge L_2 to itself;
  3. Go to step (2);
  4. Terminate
- Reachability graph algorithm properties [7]
  1. There is only one instance of a transition in graph.
  2. Due to aforementioned characteristic, the algorithm always ends.
  3. Due to ω's monotonicity, the reachability graph's nodes are a completed and are not cyclic (apart from a few self-loops).
  4. The reachability graph's node count is constrained by $\min\{|P|, |T|\}$, where P is places and T is transitions;
  5. $|T|$ limits the reachability graph's edge count.

  Below are some Theorems in Ref. [2] on how can we do verification on rule base systems based on reachability graphs.

  Theorem 1: It's worst-case computational complexity is $O(\frac{1}{2}(|P| \times |T^2|))$ [7]

  Theorem 2: It can identify missing errors. [7]

  A transition does not display as label unless it is enabled and the rules relating to disenabled transitions doesn't appeared thus displaying the missing rules.

  An example in Figure 8 shows how a rule base for R1 is transformed into a ω-net. Nodes p_6 and p_7 are not marked with ω and rules r_7, r_8 not in labels. Thus the rule base is incomplete. Rules which are in self-loop and hence have circularity. [7]

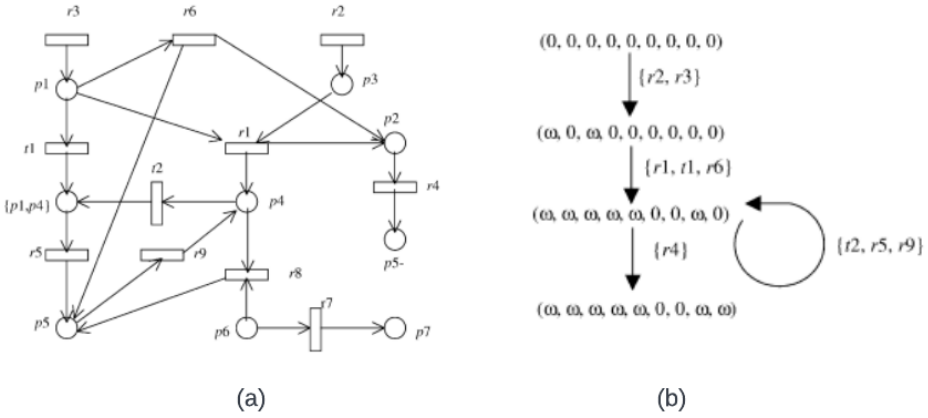

Figure 8: (a) The ω-net for R1 (b) Reachability graph for R1 [7]

4.2 Verification of Scenarios

- Use case construction:

  Identifying actors and defining use cases for them are the two processes in the construction of use cases. A person or entity that interacts with the system is represented by an actor which are system's direct users. A certain form of system use is corresponding to a use case. It is a simulation of a system's capability that is activated in response to an external actor simulation. A use case diagram is constructed from communication (participation) links between the actors and the use cases. [9]A use case figure is a collection of use cases that are constrained by a boundary of the system. The following are the four primary types of actors:

  1. Direct users of system functions are considered the primary actors.
  2. Supporting actors: Individuals who carry out administrative or maintenance duties
  3. External hardware: The necessary hardware components of the application domain that cannot be avoided and must be employed. It is relevant to the other hardware peripherals rather than the machine on which the application is run.
  4. Other systems: Additional systems that the system must communicate with.

- Scenario construction:

  Information that is transmitted and how the system uses it are stated in terms of scenarios. A two step process to specify scenarios is mentioned below: [9]

  1. Scenarios which focuses on the major behaviour of the system without taking the objects into consideration. The analyst communicates with the users using these scenarios in a descriptive form to confirm that they comprehend the system. Scenario tree as described in Figure 10 is used for this. Refer Figure below.

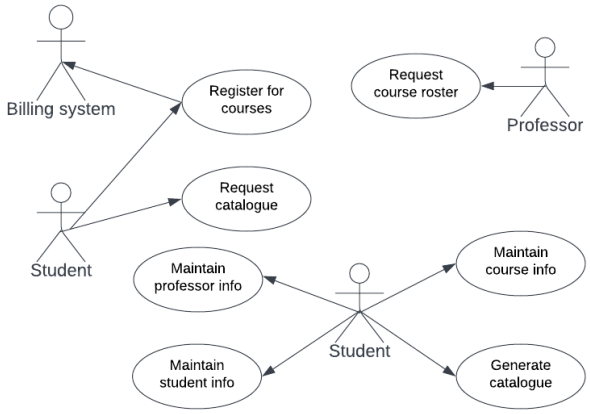

Figure 9: A use case diagram for course registration system [9]

2. To make scenarios more precise so that they concentrate on the interactions that take place between system elements over time. Sequence diagrams serve as a representation for the scenarios at this stage. Refer Figure 4

- Petri-nets construction from scenarios:

  The lifeline of an object in a sequence diagram is used to illustrate the behavior of an object in sections. In order to describe the states and events, respectively, as a result, the item's lifeline is transformed into a series of places and transitions. Three topics can be covered while talking about the transformation: object lifelines, object interactions, and temporal restraints.A sequence diagram displays the interactions among the involved items ordered in time sequence. Figure 11 is an example how the sequence diagram can be converted into Petri-net model.

  Algorithm for converting sequence diagram into TPNs: [9]

  1. Objects in sequence diagram:
     a) For m states insert m places in TPN
     b) For n diverse events insert n transitions in TPN
     c) As per the time ordering, connect this places and transitions in the lifeline.
     d) Starting place is replaced by a token. [9]
  2. For events in sequence diagram:
     a) For synchronous event, add return message in sequence diagram
     b) A dummy is inserted if an event occurred.
     c) Connect source, dummy and destination object in the same sequence of transmission.
     d) If two same message if sent to other destination object consequently, insert dummy for each transition.
     e) If an event is transmitted from the source object to itself, link another concurrent place to it else there would be no connection. [9]

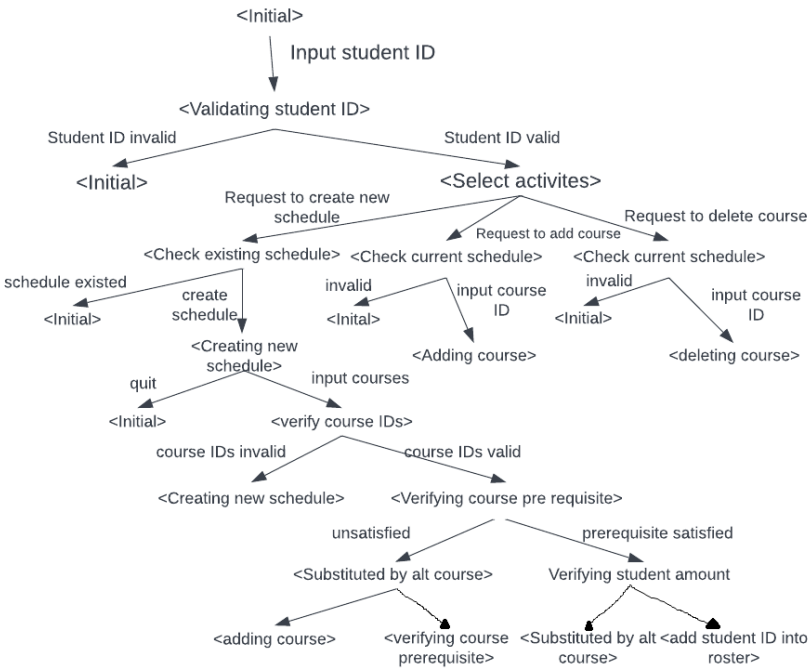

Figure 10: A scenario tree for student, registration of courses [9]

A lifeline is a series of states and transitions that an object experiences as a result of creating and accepting events. These states and transitions make it possible for the object to be changed into a collection of places and transitions, where the place represents the state of the object at a given time and state transition.Initially token is placed at initial place of object in TPN model. As the events are sequential, TPN model could be used to represent the ordering relationship. As the model executes the tokens will be forwarded to the next event in sequence diagram.In synchronous messages return message will added to constrain the sender as blocked and waiting till processing the message while the asynchronous message doesn't have any blocking and continue sending message. In concurrency different objects can generate events simultaneously. [9]

4.3 Verification of system using coloured Petri-nets

The availability of tools facilitating the generation and contrivance of models is crucial for the practical application of CPN modelling and analysis. A tool set called CPN Tools is used to edit, simulate, analyse state space, and performance analysis on CPN models. Here we will

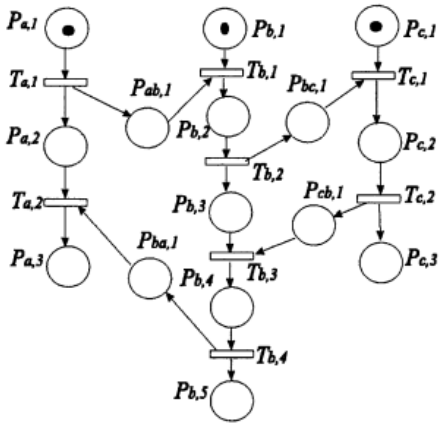

Figure 11: A Petri net transformed from sequence diagram in Figure 4 [9]

take an example of Health system. This system's main objective is to gather and manage public chronic conditions and alerts. [5]Additionally, it is utilized to alert people of critical details pertaining to the health system. A CPN model of this system explains the states of systems and events that could shift the states. Its viable analyse various scenarios and investigate the nature of the system by simulating the CPN model. Simulation is frequently used to investigate and debug system designs. [5]

Single step debugging is comparable to an interactive simulation. It offers a technique to go over a CPN model, carefully examining many scenarios and determining if the model performs as predicted. The modeller controls an interactive simulation and chooses the subsequent step from among the current activated states. [5]

The working of token system can be explained with one example for accessing the system. If token type *access* and *UserType*, the transition *Access* system is activated. Both tokens are compelling when the transition is binding, and a new token is created with the function *NextTypeUser* instead of *User* and another token with the state *System Released* in place. [8]

Simulation report generated by the CPN monitoring tools can be used to detect what happened next. The report is a contextual file containing detailed information about all bindings and transitions that occurred. Binding elements are monitored with markings when they are at simulation state. Break monitor stops after an expected condition is fulfilled. Result is secured only after 10000 simulations. [8]It will show the statistics of all access queries like how many people access the system, which kind of requests were made etc. A significant number of verification questions about the nature of the system, such as the absence of deadlocks, the feasibility of always being able to achieve a particular state, and the assurance of the delivery of a certain service, can be answered from a created state space.

A state space report shows statistics how large a space is. Next part contains information about boundedness properties, it gives no of tokens a place can store. The best upper integer bound of a place specifies the maximum number of tokens that may be present there in any reachable marking. The smallest no of tokens that could be stored in a location on any reachable

marking is specified by the best lower integer boundaries for that place. [5] [8]

4.4 Verification of use cases in Requirement analysis

The TCPN model is obtained through the interconnection of subnets. The use case description's *Pre-condition* is transformed into a subnet first, followed by the sequential transformation of the *Event flow* into subnets, and finally the *Post-condition* is transformed into a TCPN subnet.A tool named Platform Independent Petri Net Editor, which is acustomed to verify Petri nets, produce reachability graphs, is used to analyse the TCPN model for use case. [15]

5 Conclusion

In Ref[2] we have presented a new approach to find structural errors in rule base systems using ω-nets. As mentioned in Section 4.1, both the transformation of rules into ω-nets along with creation of a reachability graph can be done automatically with ease. Due to the employment of ω, the specialized graph has more impact than generic reachability graph. Low complexity of computation. Since there aren't many intermediate states produced, the state space is limited. Here, the structural errors are mostly based on information given by domain experts thus difficult to point missing rules. Using a reachability graph has the benefit of allowing rules that resulted in various structural faults to be corrected. [7]

In Ref[3] we presented verification of scenarios using Petri-nets. The verification mechanism is designed to detect any missing or wrong information occuring in the scenarios. As the method is deadlock free within sequences diagrams are bounded by time line and no integration can exist. Advantages of TPN for this approach is, it provides flexible perspective for scenario and verification monitoring can be done by doing reachability analysis on TPN but there is a disadvantage as well that it has limitations in reaching scalability of reachability analysis.Another benefit is that a CASE tool is already available to facilitate detect errors in requirement as soon as possible to cut expenses and effort. [9]

In Ref[4] we presented formal verification using coloured Petri-nets. In this the models depicting exact functionality are designed, analysed and simulated. The user just have to create the semantic model, also CPN tools are free of cost and easy to understand. CPN includes tools for automatic verification and a precise mathematical semantics. With the aid of CPN Tools, it is attainable to simulate modelled system's nature in order to examine its behaviour, to validate properties using state space and model checking. [5]

In Ref[5] we used use cases in use case diagrams[Figure 8] for requirements elicitation. Verification of use cases is done to check correctness and its validity. Experiments demonstrate that our method can assist developers in minimizing disparity and minimizing requirement analysis errors. [14] Only a few Petri net analysing techniques are used to verify the use case, hence more study into Petri net analysing techniques is required to uncover more flaws. Our study in this area is primarily concerned with the modelling and verification of functional requirements. By extending TCPN to the modelling and verification of requirements that are added on top of functional requirements, software performance will be enhanced. [15]

A lot of related work has been implemented in the problems discussed above and a lot of

future work is also expected in this field with a good scope of efficiency and robustness. Most of the work done fails to check the timing properties of the Petri-net network but able to determine the deadlocks, violation of the invariants and informal representation like textual, grammar and Automata. [9]The grammar model is translated into a corresponding state machine and after taking into consideration both the models an equivalent abstract model. All this modelling is done from the perspective of the end user. The final abstract model is then determining the verification properties of the system. While doing the verification in the requirements elicitation phase using use cases there is still lot more work that could be implemented to improve the functionality of the system. [15]One is to use more techniques to use other properties of Petri-nets to verification and other is to take care of non-functional requirements as well which are dependent or affecting the functional requirements of the system. We have specified the verification of UML diagrams in the above references, the problem with that is UML can be subjective and every time not a robust UML is presented, it might be different for other user's perspective that is it can be subjective. [5]While the Petri-nets has a strong semantics which could help in automatic verification, so this is the gap between the verification process of UML diagrams. For the ω-nets, the process could be fully automated once there are provisions to generate the rules to reachability graphs, then it can be used in real time systems. [7]

References

[1] Nektarios Georgalas Achilleas Achilleos, Kun Yang & Manooch Azmoodech: *pervasive Service Creation using a Model Driven Petri Net Based Approach.* Available at https://www.researchgate.net/publication/4366523_Pervasive_Service_Creation_using_a_Model_Driven_Petri_Net_Based_Approach.

[2] Maldoddi Praveen Alexander Heußner: *omega-Petri nets.* Available at https://www.researchgate.net/publication/235326446_omega-Petri_nets.

[3] Thong Weng Jie; Mohamed Ariff Ameedeen: *Model Driven method to represent Free Choice Petri Nets as Sequence Diagram.* Available at https://ieeexplore.ieee.org/document/7333104.

[4] Miryam Reyes Carla Pacheco, Ivan García: *Requirements elicitation techniques: a systematic literature review based on the maturity of the techniques.* Available at https://ietresearch.onlinelibrary.wiley.com/doi/10.1049/iet-sen.2017.0144,https://ietresearch.onlinelibrary.wiley.com/doi/pdf/10.1049/iet-sen.2017.0144.

[5] Michel S. Soares Franciny Medeiros Barreto, Joslaine Cristina Jeske de Freitas & stephane Julia: *A Straightforward Introduction to Formal Methods Using Coloured Petri Nets.* Available at https://www.scitepress.org/papers/2014/48619/48619.pdf.

[6] Prof. Dr. Robert Gold: *Petri Nets in SoftwareEngineering.* Available at https://www.thi.de/fileadmin/daten/Working_Papers/thi_workingpaper_05_gold.pdf.

[7] Hongji Yangc A udong Hea, William C. Chub: *new approach to verify rule-based systems using petri nets.* Available at https://www.academia.edu/10939931/A_New_Approach_to_Verify_Rule_Based_Systems_Using_Petri_Nets.

[8] Kurt Jensen: *A Brief Introduction to Coloured Petri Nets.* Available at https://link.springer.com/content/pdf/10.1007%252FBFb0035389.pdf.

[9] Jong Yih Kuo Jonathan Lee, Jiann-I Pan: *Verifiying scenarios with time Petri - nets.* Available at https://www.researchgate.net/publication/223064669_Verifying_scenarios_with_time_Petri-nets.

[10] Didier Buchs Stefan Klikovits & Alban Linard: *Pro Git petri Nets A Formal Language to Specify and Verify Concurrent Non-Deterministic Event System.* Available at https://link.springer.com/content/pdf/10.1007/978-3-030-43946-0_7.pdf.

[11] Carl Adam Petri: *Carl Adam Petri and Petri Nets.* Available at https://www2.informatik.uni-hamburg.de/TGI/PetriNets/history/CAPetriAndPetriNets.pdf.

[12] Carl Adam Petri: *Zur Person Carl Adam Petri und den Petrinetzen.* Available at https://www2.informatik.uni-hamburg.de/TGI/PetriNets/history/CAPetriAndPetriNets.doc.

[13] Krzysztof Sacha (1998): *Safety Verification of Software Using Structured Petri Nets. SAFE-COMP.* Available at https://link.springer.com/chapter/10.1007/3-540-49646-7_26.

[14] Mohammad Suaib Tabbassum Iqbal: *Requirement Elicitation Technique: -A Review Paper.* Available at https://www.researchgate.net/publication/272944882_Requirement_Elicitation_Technique_-A_Review_Paper.

[15] Jinqiang Zhao & Zhenhua Duan: *Verification of Use Case with Petri Nets in Requirement Analysis.* Available at https://link.springer.com/chapter/10.1007/978-3-642-02457-3_3.