Composability Verification of Complex Systems

Using Colored Petri Nets

By

Syed Hassan Askari

00000118585

Supervisor

Dr. Imran Mahmood

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Information Technology

Approval

It is certified that the contents and form of the thesis entitled "Composability Verification of Complex Systems Using Colored Petri Nets" submitted by Syed Hassan Askari has been found satisfactory for the requirement of the degree.

.

Advisor: Dr. Imran Mahmood

Signature: ____________________

Date: ____________________

Committee Member 1:

Dr. Muhammad Sohail Iqbal

Signature: ____________________

Date: ____________________

Committee Member 2:

Dr. Safdar Abbas Khan

Signature: ____________________

Date: ____________________

Committee Member 3:

Dr. Anis ur Rahman

Signature: ____________________

Date: ____________________

THESIS ACCEPTANCE CERTIFICATE

Certified that final copy of MS/MPhil thesis written by Mr. Syed Hassan Askari, (Registration No 118585), of School of Electrical Engineering and Computer Science (SEECS) (School/College/Institute) has been vetted by undersigned, found complete in all respects as per NUST Statutes/Regulations, is free of plagiarism, errors and mistakes and is accepted as partial fulfillment for award of MS/M Phil degree. It is further certified that necessary amendments as pointed out by GEC members of the scholar have also been incorporated in the said thesis.

Signature: ______________________

Name of Supervisor: Dr. Imran Mahmood

Date: ________________________

Signature (HOD): _______________

Date: _________________________

Signature (Dean/Principal): ________

Date: _________________________

Certificate of Originality

I hereby declare that this submission is my own work and to the best of my knowledge it contains no materials previously published or written by another person, nor material which to a substantial extent has been accepted for the award of any degree or diploma at NUST SEECS or at any other educational institute, except where due acknowledgement has been made in the thesis. Any contribution made to the research by others, with whom I have worked at NUST SEECS or elsewhere, is explicitly acknowledged in the thesis.

I also declare that the intellectual content of this thesis is the product of my own work, except for the assistance from others in the project's design and conception or in style, presentation and linguistics which has been acknowledged.

Author Name: Syed Hassan Askari

Signature: ____________________

Acknowledgment

I pay my gratitude to Allah almighty for blessing me a lot and without His guidance I couldn't complete this task. I would like to dedicate my thesis to my Parents, specially to my late father Mr. Syed Nasir Ahmad, my mother and my sister who supported me and made me see this day.

I would specially like to thank my Supervisor Dr. Imran Mahmood for guiding and encouraging me to complete my thesis. His timely and efficient contributions helped me to shape the thesis into its final form and I express my gratefulness for his sincere supervision all the way.

I am thankful to my GEC members Dr. Muhammad Sohail Iqbal, Dr. Safdar Abbas Khan and Dr. Anis ur Rahman who helped me in my research phase by providing valuable comments. I offer my tribute to the place of knowledge i.e. National University of Sciences and Technology (NUST).

Finally I would like to thank my brother Syed Ali Raza and his Family, specially his son Syed Muhammad Faiq Raza and his daughter Abeeha Batool.

Publications

Conference Publications

Imran Mahmood, Syed Hassan Askari, Asad Waqar Malik, “Composability Verification of Complex Systems using Colored Petri Nets ”, Models, 2018.
Submitted [04-05-2018].

Table of Contents

List of Abbreviation

CPN	Colored Petri Nets
SSA	State Space Analysis
SSL	static-semantic level
DSL	dynamic semantic level
CBSD	Component Based software Development
HCPN	Hierarchal Colored Petri Nets
CBD	Component based development
RS	Requirement Specification
V&V	Verification and Validation
M&S	Modeling and Simulation

List of Tables

List of Figures

Abstract

The discipline of component-based modeling and simulation offers promising gains including reduction in development cost, time, and system complexity. This paradigm is becoming quite profitable as it promotes the use and reuse of modular components for adequate development of complex simulations. Achieving effective and meaningful model reuse through the composition of components still remains a daunting challenge. "Composability", an integral part of this challenge, is the capability to select and assemble model components in various combinations to satisfy specific user requirements. Different researches in this area have given rise to the development of different component reusability frameworks. However, lack of support for composability verification makes it difficult to achieve effective and meaningful reuse. For this reason there is a need for an adequate framework to verify and validate composability of composed models and ensure the satisfaction of desired system properties. In this thesis we propose the use of Colored Petri Nets for component oriented model development, model composition and the verification of composed models using state-space analysis technique. We present a case study of an elevator model as a proof of concept. Our case study explains the proposed process of developing and composing CPN based model components, and verifying the composed model using state-space analysis. A verified composition asserts meaningful reuse of model components by satisfying given requirement specifications.

Keywords: *Model Composition, Composability verification, Colored Petri Nets, State-space Analysis, Elevator Model*

Chapter 1

Introduction

This chapter provides the opening and general information of the research to provide a clear understanding about this thesis. It also covers the problem statement along with solution statement.

A system made by composing many components which interact with each other to perform some task is called a complex system. A network can be used to represent these systems in which the nodes will represent the components and the links between them will represent the interactions between the components. Earth's global climate is the common example of complex system.

The behavior of complex systems is difficult to model because there are dependencies and relationships between parts of a system. Complex systems have unique properties due to the relationships like spontaneous order, feedback loops etc. Because these systems are present in different kinds of fields, the similarity between them is a new area for research [1].

A model component is an independent element, specified using a formalized description, conforms to certain component standard, has well-defined interfaces, and encapsulates certain behavior. A model component can be independently deployed, and it is subject to third-party composition with or without modification [2]. Component provides the solution to the modeler in making their model, while avoiding redundant solutions to the problems. Components depend on experience, as they provide a sound solution for problems that are continuously occurring in a certain domain. Modeler working in similar domain experience reoccurrence of same problem repeatedly. Sound understanding of a certain problem gives modelers a wide choice for picking best possible solution by looking up in component catalog. This consumes less effort and time in development while providing novel solution.

A software component is defined in terms of building components separately and then merging different components in order to build a large complex model. Different components can communicate with each other through input/output ports. Detail of the component is hidden from the end user as they don't need to go inside of the component. Therefore it can be easily used by third party without knowing the inner detail of components [3] [4].

Composability is the ability to select and merge various components in order to satisfy user requirements. Reuse is defined in terms of using existing component for different applications.

Colored Petri net (CPN) is a graphical modeling language for simulation and modeling, as well as for the verification of discrete event systems. CPN models are executable and are used to model and specify the behavior of parallel system. These models are both state and action oriented. Moreover, CPN tool provides components to simulate the proposed solution. It's applicable in various domains not only in work flow modeling but also in designing complex concurrent computer systems.

We propose reusability of components and verify their relationship between components and also allow easy navigation using Colored Petri Nets language. Composability needs verification. We use verification technique named as "state space analysis". Using this technique, we ensure the correctness of system property and also verify the models that fulfill the requirements. Through state space analysis we can calculate all the reachable states and changes in states during execution. These states are represented as a directed graph. Using these graphs, it's possible to answer a verification questions about the behaviors of system such as absence of deadlock, the possibility to reach in a good state, and prevent entering in a bad state. The goal is to make a component and the modelers need to search a component related to problem in a solution catalog using these components. So that they don't need to develop a model from the scratch in order to satisfy the user requirements.

1.1 Challenge

Building a large system from the scratch is a challenging issue. Even for a small system every time modeler builds a system from scratch, as it can increase the time and cost of development. Another important issue is composition of reusable components. Verification is essential to check the correctness of system.

1.2 Problem Statement

All the information and objectives/ constraints that define the problem are called problem domain in the engineering terms. The goals which have to be achieved, the context of the problem and all the rules which define the functions are included in the problem domain. The environment in which the solution has to operate is represented by the problem domain.

Building complex models through the Composability of reusable Components is a challenging task. In order to achieve the above, it is a necessary condition to verify a given composition of components, which fulfills the requirement specification, using a suitable approach.

1.3 Solution Statement

The information that is used to define the proposed or expected solution of the system is called solution domain. The concepts, laws, techniques, software architects, algorithms and recommended ideas and practices which help in finding the solution of the problem are included in the solution domain.

The solution statement is defined based on the approach proposed: "The verification of component model includes that the components should be composable at all levels and the requirement specifications are met according to the defined objectives and fulfilling required constraints."

The base components are re-usable in a correctly composed model thus rapid model development is supported and the components can also be used again or in another model.

1.4 Key Contributions

We build a component in colored petri nets tools. Once the components are made we combine them according to their requirements. Instead of building model from the scratch modelers need to search a component from repository and build a system according to their requirements. Verification is used to check the correctness of composable model. Different approaches are used to verify the system; we used state space analysis approach to verify our system. Through verifying different verification properties, we ensure that model is correctly connected and fulfills the desired goal.

1.5 Research Impact

Our proposed framework is to provide a component that will help the modeler in making models of concurrent systems. Modelers must have domain knowledge in order to find the component in solution catalog. It will help the modeler to reuse existing components. Through reusability of existing components, it can reduce the time and cost of development.

1.6 Thesis Organization

Rest of the thesis is organized in following chapters

1.6.1 Chapter 2: Background

Chapter 2 provides brief overview of model components, model composability and verification of model explains its impact on real life. This chapter also explains CPN tools which can be used to simulate our elevator model. Moreover, few preliminary concepts, used in methodology chapter, have also been discussed.

1.6.2 Chapter 3: Literature

This chapter explains the work done so far related to model components, model composability and verification of model and summarizes the hierarchal approaches of model component and model composability and also formulates research directions for this dissertation.

1.6.3 Chapter 4: Methodology

Our proposed component based model reuse has been presented in this chapter. As we built component library in which modeler can build complex model using existing components. We also verify the composed components in order to check the correctness of model. Furthermore, working of different verification properties is used to check the model's accuracy that has been discussed.

1.6.4 Chapter 5: Composability Verification of an Elevator Model

This chapter is dedicated to demonstration of the functionality of our proposed framework. A case study of Elevator model has been used to provide the proof of concept. The chapter is concluded by results and their detailed discussion.

1.6.5 Chapter 6: Conclusion and Future work

The work accomplished in this thesis has been concluded in this last chapter. Moreover, the chapter also describes the future directions which can be done ahead to this work.

Chapter 2

Background

In this chapter we discussed the information about the topic which is being studied. It describes the definitions and concepts of model components and model composability to get understanding about this thesis.

2.1 Complex System

According to [5] complex systems consist of large connections between components where there is no central control and complex collective behavior arises due to simple individual rules. Thus, the collective behavior of a complex system arises from the simple individual behavior of its parts. The emergent behavior is the property of a whole system rather than of isolated parts of the system. For example, consciousness is the collective property of the whole human body not of the isolated neurons. Moreover, complex system can still manage to work even if some of the parts of the system are removed because they adapt to the changes [6]. The properties of a whole system can be called robust when it is compared with respect to the loss of their parts. Lastly, the emergent properties only rise when the internal components of the system interact with each other. The combination of related multiple subsystems which are organized in a hierarchy is known to be a complex system see **Figure 1**. The subsystems have to work together to get the required output [7]. The same model which enforces working of components together is used throughout the system. It is feasible for the modeler to break down a complex system into subsystems according to their functionality.

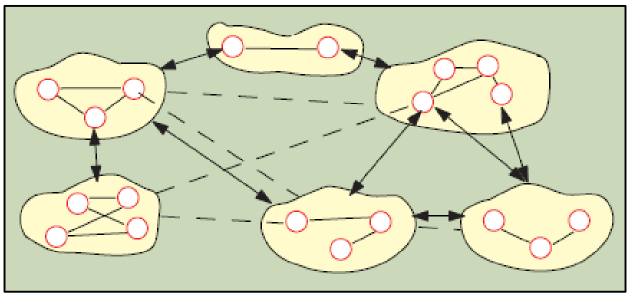


Figure 1: Complex System

In other words, it can be said that each subsystem or component has a specific functionality. Moreover, the complex systems do not merely have any top which means their functionality totally depends on the components being used in them [8]. The individual components should be local and encapsulated for its own functionality. That means that the components should be active and they should be able to control their own actions.

2.2 Model Components

A Inspired from the discipline of *Component based Software Engineering*, models are built as building blocks and are referred to as "*Model Components*". A model component is an independent element, specified using a formalized description, conforms to certain component standard, has well-defined interfaces, and encapsulates certain behavior. A model component can be independently deployed, and it is subject to third-party composition with or without modification [2]. A set of model components are composed together to create complex models. The model components can be composed if their interfaces match each other. However it is difficult to say whether they will work together in a meaningful way, and will fulfill desired requirements unless their composability is verified. The predictability of guaranteeing the correctness of model composition is called Composability.

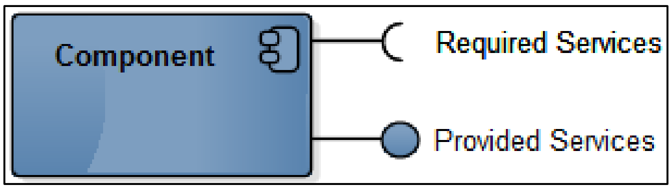


Figure 2: Component

2.3 Component Reuse

Software community faces different issues that are elevated with the help of fast and cost effective development. To overcome this problem component reuse approach is used which promises effective development of systems as it can reduce the development cost and time. In component based development different application can be developed by using existing components rather than a monolithic entity. A single component can be used in many applications. Using component reuse approach can reduce production cost by integrating the existing components instead of building new components from the scratch. It will help the modeler to reuse existing components according to their requirement. In order to reuse component, we need Composability [9]. Hence, verification is required in order to achieve successful Composability.

2.4 Composability

Composability is the ability to select and merge various components in order to satisfy user requirements [10] [11]. Reuse is defined as using existing components for different applications [9] [12] [13] [14]. If the components are not composable then by definition their composability is not valid. Verification is required in order to achieve successful Composability [2]. **Figure 3** illustrate that model A and model B can be built by composing different components. modeler can get components from component library and build a model according to their requirements.

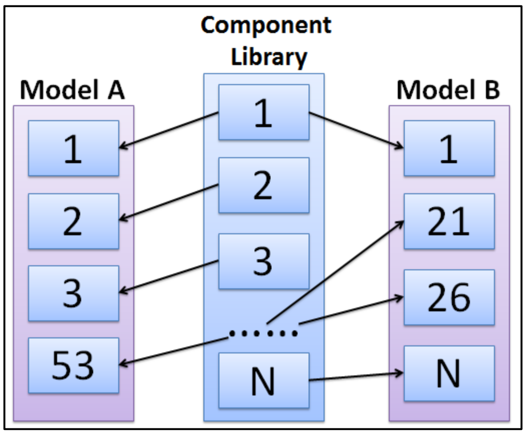


Figure 3: Composability

Two basic types were emphasized upon by Petty and Weisel which are syntactic and semantic. They proposed the importance of these composability models in their theory of composability. The composable components in syntactic should be built with companionability like assumptions of timings, and passing parameters. The syntactic composability deals with the connectivity of components i.e. either they can be connected or not. On the other hand, semantic composability deals with the complete model i.e. if the models are capable of forming a complete simulation system and the composed model is valid according to semantics or not. It can be possible that two components may be semantically invalid but they can pass parameters to others. Different research group use different levels to understand composability in depth. To understand the levels of composability, different levels of agreement are required between them. Five distinct levels were recommended by Davis which are syntax, semantics, pragmatics, assumptions and validity. These levels can also be defined as different composability consistencies which are observed in order to ensure model composability correctness. Nine levels are suggested by Petty and Weisel when composability is studied in terms of a combination of composition units. Levels of conceptual interoperability were described by Tolk to study the functionality of composability and interoperability. This model consist of six layers named as technical layer, syntactic layer, semantic layer, pragmatic layer, dynamic layer, and the conceptual layer. Similarly, a four layer composability checking model was introduced by Medjahed and Bouguettaya which checked the web service composability. **Figure 4** illustrate different level of composability which are: Syntactic, Static Semantic, Dynamic Semantic and Qualitative level. These levels are considered basic benchmarks for checking model composability. The idea of composability of models is strictly dependent upon the levels' consistency as explained further.

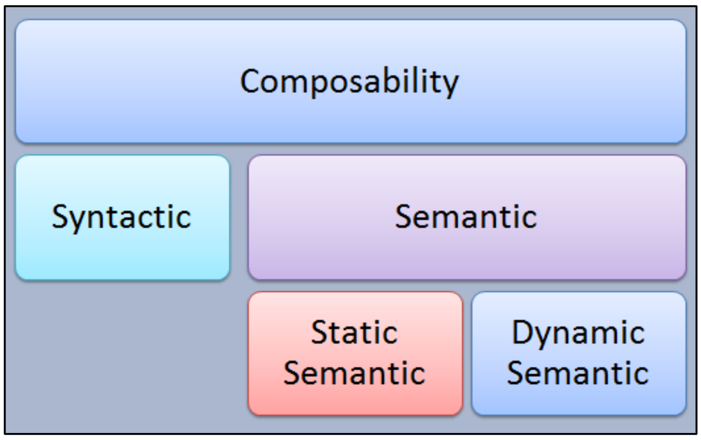


Figure 4: Composability Level

2.4.1 Syntactic level

The structure of components is studied in this level to check whether they will fit together or not i.e. one component's output is given as input to other component and it is checked if the syntactic information of under consideration components match with each other or not i.e. message, action mode and total parameters.

2.4.2 Static-Semantic level

Meaningful connection of components is checked on static-semantic level (SSL). It consists of having a complete and accurate understanding of the information exchanged by the components involved in the model being composed. The same understanding of the terms and messages is ensured on this level so this level consists of exchanged information's same meaning between components. For example, if two components are assumed to process a value that is given to them by considering the value in integer but one of the component processes the value as float or character then the composability can result in a situation unwanted by the user. The following example explains the level: If the distance has to be measured in degrees but the component interprets it in a different unit then it will be a semantic mismatch. The term static is used in this level because the values or information checked here remains constant and does not change during the whole model composability.

2.4.3 Dynamic-Semantic level

Dynamic consistency is ensured on dynamic semantic level (DSL) i.e. the components have exactly the same behavior which is required to reach the desired goal. Behavioral consistency and coherency is ensured to achieve the system model goals in dynamic level composability. It can only be reached if the components work properly according to the states and transitions. And the components should have a behavior according to the requirements to collectively make progress i.e. two behaviors are expected from a waiter component in a restaurant model. One of them is that the order is taken by the waiter, food is served and payment is collected and the other one is where the order is taken by the waiter, payment is collected and food is served after preparation. The composability of model will be affected by correct customer component and correct behavior of waiter. A customer who is expecting the first behavior at a restaurant of second type will have to wait for the food forever. So in order to make a model achieve its desired goals, correct dynamic behaviors are necessary.

If the behavior is not right but the components are at their right state, the model goal may not be achieved. e.g. Two machine components are produced by two machine components in a manufacturing system which have to be joined later and a single robot component is shared by both for input of material and for manufacturing. The robot should be just providing equal chance to both machines to process. The final result will be unbalanced if the robot is not just and the system will be deviated from its goal. The term dynamic is used because the states and behavior change during the model composability and components' interaction.

2.5 Verification

Verification is defined as a process that can check model implementation accurately [15] according to the specifications [16]. We use verification technique named "state space analysis" to check the system properties and verify if models fulfill the requirements [17] [18].

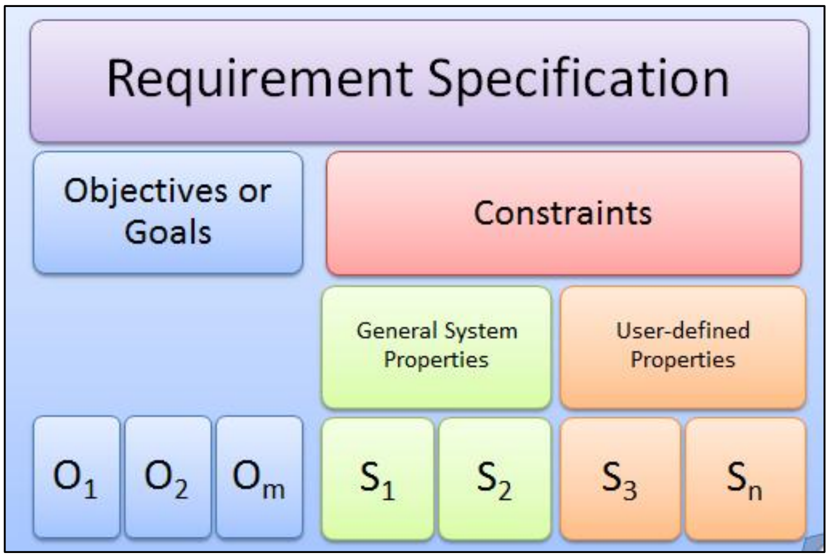


Figure 5: Requirement Specification [19]

Verification is concerned with building the model correctly in M&S. It is the process of implementing the model right [15] and whether the system is according to the specifications or not [20]. Verification deals with the correct translation of the requirements into the model and the model should be executable which is the basic principle of M&S. An abstract description of a real system [21] is a conceptual model which is made on the basis of requirements and modeling purpose**s**. The conceptual model is then refined into a more concrete executable model [22]. A model design's subset can also be called conceptual modelling. Conceptual modeling includes moving between problem situations where model requirements define what should be modeled and does not depend on its implementation details which are later dealt with in executable model. **Figure 5** illustrate the modeler gathers the information and formulates the requirement specifications.

2.6 Colored Petri Nets Tools

Colored Petri net (CPN) is a mathematical modeling language for simulation and modeling, as well as for the verification of discrete event system. Essentially, it is a powerful language that can be used to display parallel or concurrent activities in a system [23]. Using CPN tool [24], a model is created for analyzing system performance. Colored Petri nets are a buildup and well-researched means for system modeling and simulation [25]. A Colored Petri net is a bipartite diagram in which we have places, transition and arc. Arc runs from transition to place and vice versa, but never between transition and between places.

2.6.1 Formal Definition of Colored Petri Nets

A colored Petri net is a nine tuple: **CPN** = (**P, T, A, Σ, V, C, G, E, I**) where:

1. P= {p1,p2,…,pn} is a finite set of places
2. T= {t1,t2,…,tm} is a finite set of transitions such that: $P \cap T = \emptyset$
3. $A \subseteq P \times T \cup T \times P$ is a set of directed arcs.
4. Σ is a finite set of non-empty color sets.
5. V is a finite set of typed variables such that: Type[v] $\in \Sigma$ for all variables $v \in V$.
6. C: P→Σ is a color set function that assigns a color set to each place.
7. G: T → Expression is a guard function that assigns a guard to each transition
8. E: A→ Expression is an arc expression function that assigns an arc expression to each arc.
9. I: P → Expression is an initialization function that assigns an initialization expression to each place p.

Detail explanation of colored Petri nets is beyond the scope of this paper therefore interested readers are recommended to see [23] [24] [26]. CPN is a well-known tool for the development of models, their verification using state-space analysis and analysis of CPN models [24]. It provides integrated development environment (IDE) for the development of CPN models. For efficiently executing timed and untimed nets, it has a bundled simulator. State-space analysis is the most important feature of CPN tools. It consists of different query functions and helps to create analysis report which plays an important role in the verification process.

2.6.2 Elements of CPN

Colored Petri nets is a graphical language that is used in making model of concurrent systems and their verification. A Colored Petri nets consist of **Places**, **Transitions** and **Arcs. Table 1** illustrates the elements of colored petri nets.

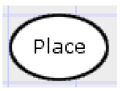
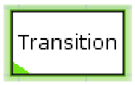
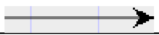
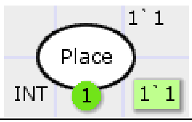
Elements	Description
	Circle represents the places and also explains the memory of the system.
	Rectangle represents the transitions that tell the action of the process.
	Arrow represents the arcs that tell the state of the system changes when transition is fired.
	Set of tokens is assign to each place, that carries tokens of different data types like integer, string etc.

Table 1: Elements of CPN

Using these elements, we build a queue component. **Figure 6** illustrate we store the number of objects in a queue. We can add or remove object information from a queue. Different types of queue can be used for adding and removing objects in a queue e.g. Priority queue, Random queue, FIFO queue and LIFO queue. Using these queues, we specify the order of objects in which they execute.

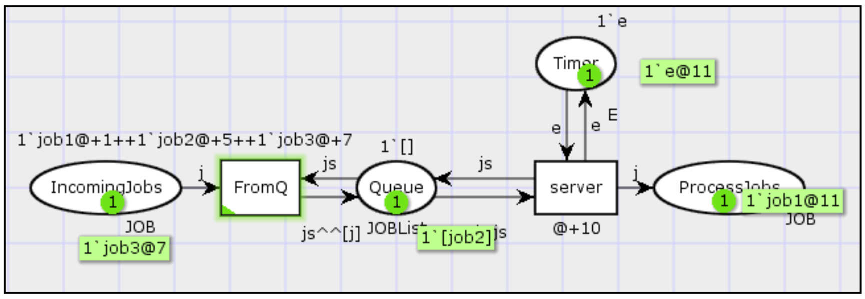


Figure 6: FIFO Queue Component

2.7 Hierarchical Colored Petri Nets

CPN consists of different modules and the modules can be considered as black boxes which perform all the processing on abstract level by concentrating on one module at a time. **Figure 7** illustrate Hierarchical colored petri nets in which entire CPN model is replaced by a transition that can be connected to the main model to make hierarchical CPN models using CPN tools. The entire model then can be broken down into different smaller modules and can be used to help in modular development [23] [24]. Colored Petri nets hierarchical features are used in our CPN component model.

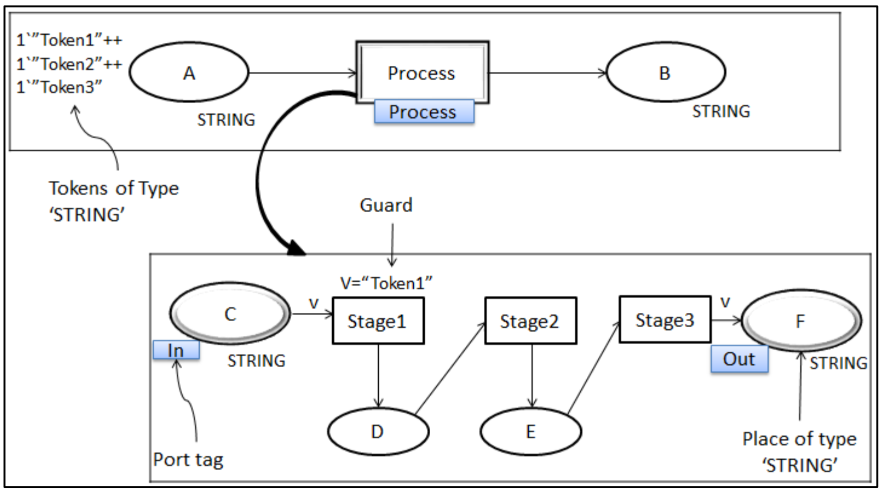


Figure 7: Hierarchical Colored Petri Nets

2.8 State Space Analysis

One of the most noticeable methods for formally analyzing and verifying system is state space analysis. All possible system states are found in this method and are represented as vertices of a directed graph while the transitions of system from one state to another are represented by directed edges. Theoretically, state space consists of nodes which represent each reachable state of the system and the arcs represent the transitions. A formally built state space graph can be used to answer a lot of questions about verification of system considering how the system behaves in situations such a no deadlock, reaching good state, never reaching a bad state and the surety that the system reaches its goal state(s). Using CPN tools we can analyze state space step by step where they are located at [27]. Logical reasoning about why a certain property does not hold on a system can be provided by state space methods [26]. **Figure 8** illustrate state space analysis of a system**.** Even for a small system, there may have infinite number of reachable and goal states. This problem is increased when time is also being considered in the system. A lot of research has been done to discover methods to get rid of this problem. The full state space is not taken under consideration in reduction methods [26]. When the model becomes large, these reduction methods are one of the famous methods of reducing the problem of state explosion.

More projects are being done on concurrent and distributed systems. There are a lot of examples available which include large systems in telecom and World Wide Web technology based application or embedded systems which may also be called small systems. A concurrent system has multiple communicating methods which are independent from each other. Thus, the system execution may be done in different ways i.e. it may depend on the loss of messages, the process speed and the times on which the system receives input. This makes the concurrent and distributed systems complex whose designing is also complex. So the complexity has led to development of processes which take computer aid to analyze, validate and verify the execution of such systems.

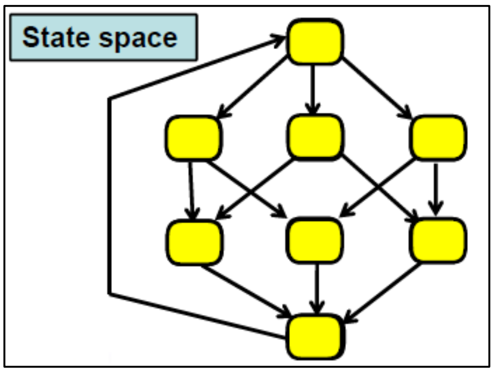


Figure 8: State Space Analysis

Some of the most commonly used methods in this field are state space methods. Computing all reachable states and changes of the system and representing them using a directed graph is the main idea of this system. The visual representation of state space using directed graph makes this possible to study the behavior of the system e.g. verifying processes of system, studying different properties and checking errors in the system. The main disadvantage of this process is state space explosion: even small systems may have unlimited number of reachable states and this is the most important limitation in the state space analysis. Therefore, to reduce the complexity involved in this process, reductions methods are the main topic of discussion. The representation of entire state space is avoided in this method or is represented in a compact form. The reduction does not affect the properties of the system and the reduced state space can be used to derive properties of the system.

The CPN tools have the State Space tool integrated which means that the user can easily switch between different tools like the editor, simulator and the SS. A state space node can be inspected in the simulator after it is found. The marking can be seen directly on CPN model's graphical representation. The enables transition examples can be seen, investigated and simulations can be made. Similarly, a marking in the simulator can be added to state space as new node.

There are a number of built in standard queries present in the State Space tool which can be used for investigation of the CPN properties including reachability, liveness and fairness [28]. A number of powerful search facilities are also present in the CPN to formulate our own queries based on the criteria we want to search on. No programming is required in standard queries. 2-5 lines code is needed in case of non-standard queries. Different properties can be verified through query function. Some of the properties are listed below.

2.8.1 Reachability

This property tells that there exists an occurrence of sequence from the marking of initial node to the marking of next node. Using state space graph we investigate the sequence between two nodes. Through query function we check the sequence between nodes. Reachable (5, 3), returns true if there is sequence between marking M5 (node 5) to the marking M3 (node 3).

2.8.2 Liveness

Dead marking tells that whether there exists a marking of dead node or not. Through query function we check the dead marking. Deadmarking (8), return false this means that M8 has some enabled binding elements.

Chapter 3

Literature Review

This chapter helps in explaining how similar is the work with that of others and how much it varies from them. Moreover, it contributes in the understanding and development of the area of research.

Table 2 summarizes the literature work into separate categories whereas details of these papers are described later.

Author (s)	Category	Paper Description	Key Features
[29]	**Software Component**	Describe the component models and classify them into taxonomy	Discuss the technology named as cornerstone of component based development
[30]		Explained the reusability of software components	Different architecture of component based development has been discussed and also explained benefits of reusing components
[31]		Explained the domain specific components framework.	Described the development of domain specific components and its impact on component based development.
[32]		Explained the applications of CDSE for the development of university ERP.	Describe about the design and development of a tailor made ERP of Nigerian University.
[9]	**Model Composability**	Describes heterogeneous complex system.	Discuss the concepts of model composability.
[10]		Focus on Composability verification.	Discuss about the reusability framework such as BOM.
[18]		Focus on the composability of real time systems	State space analysis is used to verify dynamic-semantic composability
[33]		Describe how reusability and composability can be achieved by using conceptual model	Explained the important characteristic of reusability and composability
[34]		Explained different level and types of composability	Elaborates the difference between syntactic and semantic composability and between composability and interoperability
[15]	**Model Verifications**	Focus on VV&A activities that are described in the modeling and simulation life cycle	Fifteen rules are introduced to help the researchers and managers better know what VV&A is all about
[16]		Give an introductory tutorial on verification and validation	A set of examples of verification and validation are presented
[35]		Explained the Design, Implementation, Integration, Qualification, Production, Use/Maintenance	Discuss about the verification, validation and testing
[36]		Explained the techniques which improve the quality of software products	Software quality is the combinations of several factors

[37]		Discuss about the UML and SysML	Elaborates the quantitative and qualitative techniques that combine automatic techniques
[38]		Discuss about the formal verification solutions	Provide in-depth understanding of scalable SAT-based verification techniques
[39]	**Colored petri Nets**	Introduce the petri nets	Focus on the nets and Petri created his major scientific contribution
[40]		Explained the petri nets tools	Comparison of different petri nets tools in different aspects
[41]		Discuss the different petri nets design pattern	Focus on patterns and components
[42]		Discuss the workflow modeling that represents the sequence and cooperation	Suggests a process for modeling the multi-task workflow with high concurrency
[43]		Described the CPN patterns language	Focus on interplay between control flow and data flow
[44]		Explained the composition of nets	Present a variety of case studies
[45]		Elaborates the extension of classical petri nets	Present some new features that support the CPN tools 3.0

Table 2: Literature Review

3.1 Model Component

Component based development (CBD) can be used in various application in order to build a system. Component is constructed to solve a particular problem that is occurring during development. A component is a separate part of a system that has complete functionality and well defined interfaces and also encapsulates certain behavior. Component is just like a pattern that forces the modeler to reuse existing component instead of making new component from the scratch [30].

Basha et. el [31] discuss about the comparison of repositories which enables us to select the most suitable one from the available ones for component extraction. Specific advancements are required in Domain Specific Modelling and Domain Specific Architectures to achieve a positive future. It is important to work more on Domain Specific Services as not much work has been done on this domain and only three metrics are discussed. It is possible to create components that are domain specific and the effort can be reduced in cost and time by making reusable domain specific components. A framework is needed for reusability so that multiple and different components may be extracted from the same repository. The framework proposed is to make the generic domain specific components and the work can then be extended to make a tool which allows the users or modeler to extract components for multiple domains from a single repository. The CBD-Arch-DE approach is used to obtain the proposed results which is said to be the best method in CBSD and Domain Specific Engineering. Software architecture approach is also discussed in the paper which is basically good for reuse. As our work mainly deals with reusability of components so we have our main focus on Intelligent Software Architecture Reuse Environment.

Lau and Wang [29] have explained Survey of software component model. The authors have tried not to use a single terminology in almost all ideal CBD scenarios. They, intentionally, did not use any term from a specific single component. The terminologies used by them are according to the scenario. For example, they have used the words “builder” and “assembler” for the same functionality in different phases. If they only use a singular terminology in all phases it wouldn’t have been unified term.

A taxonomy is proposed by generalization of components of software models that have been examined by the authors previously. Categories of existing components with respect to the desired data of CBD are clearly explained in this taxonomy.

There is no idea model developed till now. Composition will be possible in both phase, design and development in an ideal model by using repository. Components that are easy to use and operators enabling systematic composition will be used in this kind of model. Encapsulation and compositionality characteristics must be one of the main characteristics of such models.

Design and development of a tailor-made ERP is discussed in this paper for a University of Nigeria. University admin and academic staff and students are the target users [32]. A customized ERP was designed using CBSE approach which was reliable, affordable and adaptable. The authors have made two contributions in the ERP: First, a commercial software was developed which has practical value for the university. Second, the research community was provided with a new case-study for the application of CBSE which will add to the research already going on. In future, their important module of e-administration using CBSE can be used for research.

3.2 Model Composability

The purpose of this paper is to extend the work done by Balci – Ormby approach which will achieve R&D, notation of good benefits and increase in effectivity of complex M&S design and development [33]. The purpose is explained in different terms through:

- The advancement in the concept of R&C is explained with a component model within development of application.
- Several difficulties occur in R&C within M&S applications. Identifying and addressing these problems and considering the levels of modelling granularity is also addressed in the paper.
- Challenges of R&D are explained at different abstractions of models.
- An application of Balci – Ormsby approach is discussed in terms of R&C with a CM in ERM.

For developing heterogeneous complex systems, model composition is considered important and they can also be used to explain the structure and behavior of system in simulation models. Model composability concepts are discussed in this paper in terms of modelling formalism [9]. These composability approaches are discussed along with their challenges in terms of semiconductor supply chain manufacturing systems.

The authors present a categorical breakdown of model composability approaches. They describe each model's composability approach formulation with respect to modeling formalisms [9]. A multi-layer modeling vantage is used to describe the approaches which highlight each formalism's importance. A simplified example is used to elaborate how model composability is affected by the formalisms. Semiconductor supply chain is the most suitable example for this. The impact of the choice of model composability and varying model composition degrees, limitations and complexity of different composite models is described by the separation of specifications of model and algorithm execution.

Petty and Weisel [34] explained the working on modeling and simulation, composability is an important term. The meaning of composability differs depending upon what or how components are being joined together. Being aware of these levels of composability is beneficial as well as knowing the difference between semantic and syntactic composability is important.

Mahmood, Ayani, Vlassov and Moradi [10] proposed a process which is used to verify BOM based component at dynamic semantic levels. The term Composability means to select and merge segments in various combinations to fulfill client requirements. In order to solve the problem using existing model, it can help modeler and also reduce the development cost and time. To check the correctness of compose model verification is required. A verification technique state space analysis is used to verify the compose model.

A process of verification of dynamic composability was introduced in this paper having a major focus on the composability of real time systems. Base object model is used as conceptual framework and for executable modeling standard, CPN is used. The extension of BOM to E-BOM is suggested, so that the necessary behavioral details are present in it which are important for implementation and for time function [18]. An automatic transformation method is provided to change E-BOM into CPN component model with time functions. Representing the Model component in CPN language is useful because the original structure and behavior do not change at all in this language. A verification template is suggested for the verification of dynamic-semantic composability which can use State-Space analysis to verify the considered model.

3.3 Composability Verification

The definition of verification explained by Department of Defense Modelling and Simulation office is defined as the process of making sure that the modeler's concepts are accurately represented by the implementation of model [35].

Generally, verification (V) is a process to check consistency of a product with respect to the requirements and specifications [16]. The correctness of model is dealt with verification and helps in building the model correctly i.e. a model that has no errors and works correctly according to the requirements [15]. The accuracy of converting the model's requirements into a conceptual model and then the prepared conceptual model is converted into an executable model [16].

The terms used in model definition are defined below:

Correct: Anything with no errors, in consistency with the facts, reason, anything that conforms to the already made standards.

Correctness: The accordance of model, system or product with respect to its specifications [46].

How a system works and is it according to the defined requirements [36].

A model is said to be correct if its behavior matches with the specifications defined by the user or modeler. Hence, the specifications are a major source to check correctness of a model. A software is said to be correct if it complies with the following three aspects: i) When it has been made according to the specifications ii) When it has been developed incorrect against its specifications then it is defective and iii) When the correctness cannot be checked according to some specification then it will be unknown. The sum of all passing unit tests is the specification of software entity. Specification can also be defined as property constraints and set of goals that the composed model must perform [47]. Further, verification techniques can be classified into four categories as show in **Figure 9**.

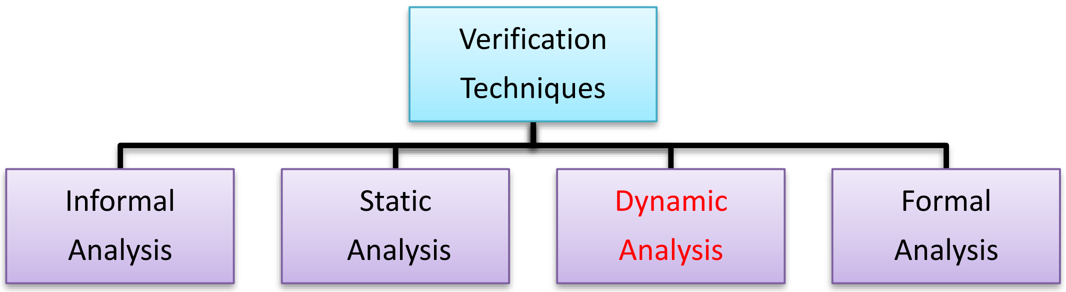


Figure 9: Verification Techniques

3.3.1 Informal Techniques

The most frequent techniques are informal techniques. Due to the methods dependency on human reasoning and no use of mathematical formalism, they are called informal [15]. There are well structured techniques and proper guidelines and standard policies are defined for using them. However, these techniques are vague and do not provide more effective result [37].

3.3.2 Static Analysis

These techniques are used for the assessment of static design of model and the implementation of model without the execution of model. The aim of this analysis is to check the structure model, the flow of data, syntactical accuracy and the consistency.

3.3.3 Dynamic Analysis

The model is executed to evaluate the behavior of model of model in dynamic analysis. The model is not only examined with respect to its execution but also the behavior is checked during execution. An additional code is added to the model's code to check the behavior of model during its execution. The additional code is called instrumentation [37].

3.3.4 Formal Analysis

The mathematical analysis of a system with respect to its unambiguous property is defined as formal analysis. These methods are called formal analysis and the properties that are checked are called formal specifications. Complete coverage can be provided on a model of the system by using forma verification. These methods use finite state machines, PN or any other specification formulation method. However, formal verification only checks the model's correctness with respect to the provided [38].

The V&V applications whole life cycle is considered really important for completing M&S research on large scale. The sponsor of the M&S effort and the institution which is conducting M&S must understand this point. The funds should be given by the sponsor under a contract and the V&V should be applied by the contractor throughout the whole cycle [15]. Application of V&V methods throughout is time consuming and expensive. The contractors usually sacrifice the V&V and documentation due to the time consumption pressure in M&S. For alleviating these problems, the modelers require computer-aided assistance. More research can be done to introduce automation in applying the V&V techniques.

Verification of BOM based models at the dynamic semantic level is considered in this paper. An extension to the standard BOM is proposed to study important behavior details and to convert it to a model that is executable like CPN [10]. Furthermore, an automatic conversion method is proposed to convert E-BOM into CPN so that a model can be checked in its executable form while its original structure and behavior do not change. After the transformation of all components, all of them are assembled using CPN hierarchy tools and the model is analyzed using state-space analysis. For verification, the modelers are required to make and verify three types of properties: System properties, goal reachability and properties that depend on the scenario. A case study of Field Artillery is discussed and its counter example is provided to show how the framework helps in verification of a given composed model at dynamic semantic level.

3.4 Colored Petri nets

Carl Adam Petri proposed a universal useful scientific method for explaining relations among circumstances and procedures (petri, 1962). He presented his work between 1960 and 1962. At that point, he produced an impressive result about nets. Those result were accepted in research [39], a few were represented in United States.

Petri net is a mathematical modeling language for modeling, simulation and analysis of discrete event system. Basically it is a powerful language that can be used to display parallel or concurrent activities in a system [40]. Using Petri nets tool, a model is created for analyzing system performance. Petri nets are a buildup and well-researched means for system modeling and simulation [41]. A Petri nets is a bipartite graph in which we have places, transition and arc. Arc runs from transition to place and vice versa, but never between transition and between places.

Workflow modeling [42] represents the sequence and collaboration that gives pattern implementation by computer. One of the famous problems is Parallel workflow pattern that is discussed in workflow modeling. Workflow uses Petri nets as a design language to build complex and parallel system. Hierarchical Petri nets are also discussed in this paper in which different segments need to perform a task independently. Subnet that starts and ends with a transition, while replacing transition with the complete subnet, the general system is also a valid Petri nets.

3.4.1 Pattern Language for Colored Petri nets

As we discussed earlier about colored petri nets so far there isn't any organized structure and groups of patterns for colored petri nets. There are many patterns used in Colored Petri Nets. Main focus of the pattern is on the interaction between data flow and control flow [43]. Purpose of pattern language is to help modeler to construct their model efficiently. Modelers don’t need to build a new model. They must understand the nature of the problem and then find the solution of the problem in the pattern catalog. Using existing solution of a problem rather than building a new model, can save the modeler time and cost.

Reisig [44] describe that the idea used in composition of Nets is an interface that serves as crossing point between two Nets joining them together as whole. Petri net basically provides an interface for two meshed networks to work together as one, thus giving required output. The nets initially are built separately each implementing its fundamental fragments and along with process continuation, are then combined together to get an effective outcome. The interface 'net' defines a model for behavior of a particular phenomenon with a sequence of work flow. This provides ease in understanding an interface working of a complex model. The arrangement of interfaces is assembled in an altering manner (so-called as commutative) and elements between interfaces are shared among each other. Along with this the order of composition of nets can also vary.

Colored Petri nets expand the classical Petri nets formalism with information time and hierarchy [45]. Using these extensions, we build a complex model as CPN. CPN, a powerful tool supports the design and analysis of these processes. Modeler needs to understand the problem domain. Inexperienced modeler may design a pattern that is unnecessary. We have a set of patterns using which, modeler need to learn how to solve a typical design problem in terms of CPN.

3.5 Our Position in the State of Art

We build a component that helps the modeler to reuse existing component instead of building new component from the scratch. We use hierarchal approach that act as a black box in which detail of the component is hidden from the user. Moreover, our solution proposes the use of state space analysis technique for verifying dynamic composability of components. State space analysis has been appreciated in CPN community for generic verification but has never been used for Composability verification.

Chapter 4

Methodology

In this chapter we proposed solution to component based model reuse in Colored Petri Nets. All components are integrated into a unified framework refer to as: composability verification framework.

4.1 Component Composition and Verification

In this section, different techniques for proposed Composability verification are discussed in detail. These methods are referred to as CBM&S cycle's building blocks and their connection will be with different phases [48] [49]. For reading Composability verification, these details are compulsory.

4.1.1 Static Analysis

Two types of static analysis are proposed by the authors. Syntactic matching and semantic matching [11] [34]. The Composability at static-semantic and syntactic level is evaluated by using these procedures. They are pre-defined rules and no execution is required and the data on which these are applied is also static.

4.1.2 Dynamic Analysis

The behavior of conceptual model is evaluated using dynamic analysis. A component matching process is performed on components for evaluating the consistency of behavior. After the successful evaluation, the in-depth verification at dynamic-semantic Composability level is performed. One of the proposed dynamic analysis techniques is chosen, as the execution is required at different levels that is why these analyses are called dynamic **analysis**.

4.2 Requirement Specification

The information is gathered by the modeler in this step and the modeler then outlines requirements using our proposed template.

$$\mathbf{RS = \langle O, S\rangle}$$

Where: $O = \{o_1, o_2, o_3 \ldots, o_n\}$ is a set of objectives

$S = \{s_1, s_2, s_3 \ldots, s_n\}$ is a set of constraints

4.2.1 Objectives

In modeling terms, an objective oi ∈ O is defined to represent certain "Reachable final state(s)" of the components within a composition, an aggregated desirable output or an emergent effect produced by the composed model which cannot be produced by individual components. In software engineering terms, an objective represents a functional requirement of the composed system.

4.2.2 Constraints

In modeling terms, a system constraint sj ∈ S is defined as a property that must be satisfied; for instance a desirable state which must be reached, or an undesirable state which must never be reached during the execution. System constrains can be defined in terms of general system properties such as deadlock freedom, fairness etc. or scenario specific properties. The notions of constraints are different from the Objectives, because they are the kind of requirements which are not the ultimate goals of the system model but are necessary conditions to achieve the desirable goals. They can be considered as non-functional requirements from the software engineering point of view.

4.3 Development & Composition of Model Components

Individual components are developed by the modeler using CPN tools. A CPN component consists of i) Colors which represent structured data-types also called places ii) tokens: Colors of different places are represented by tokens iii) Transitions consisting of actions, time delays and events iv) arcs which use arc variables to transfer tokens from one place to another (i.e. from one element to another) and communication ports are also used to connect the components for input and output. All these four parts are combined and a functional model component is formed. The individual components are then connected to each other by communication ports.

4.4 Verification of the Composed CPN model

The state space analysis is performed in the next step. CPN state space calculation tool generates state-space of the composed CPN model. The state space can be explored by using different query functions for various verification purposes. Query functions can be defined as steps to explore state-space graph. Properties of Petri nets can be verified by using these algorithms and these are based on theory concepts of colored Petri nets. A system property in requirement specification is translated into PN property. The literature has many methods and proposed approaches on how to efficiently translate requirement into PN property. In CPN state-space analysis, existing process or algorithms can be used for the development of query function for their respective properties. CPN tool also has some built in functions for tasks related to common queries.

4.4.1 General System Properties

For verifying system properties like no-deadlocks, starvation or time synchronization, state-space analysis is beneficial technique. The modeling objectives determine the choice of one of these properties as verification criteria and for the correctness of the composition, the fulfillment of these conditions becomes necessary. The property has to be specified in CPN terms and a query function has to be defined for checking its satisfaction or violation. The deadlock freedom property can be defined as there are no outgoing arcs marking in the whole state space graph. A function ListDeadMarking () is used to extract all no-outgoing markings. Empty list means that there are no deadlocks in model.

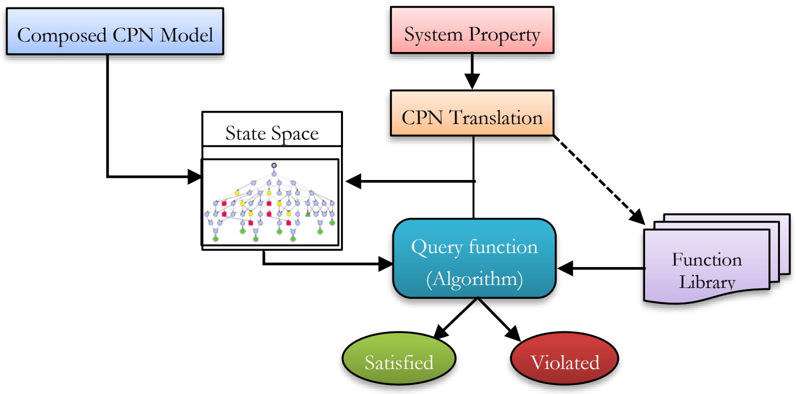


Figure 10: CPN State Space Analysis

4.4.2 Scenario Centric Properties

Safety assumptions that are particular to the scenario are proposed to be designed. Desirable situations are represented by safety assumptions. These properties might be necessary conditions but they are not the ultimate goal. **Figure 22** illustrate the working of safety algorithm which calculates all the nodes through SearchNodes () function. We have two components Door and Motor in which we have multiple places such as RotateRight, RotateLeft, EnteredF1, EnteredF2 and so on that contain tokens. These values are stored in different variables. Using if condition we ensure that either we have tokens in EnteredF1, EnteredF2 and so on places or we have tokens in RotateRight and RotateLeft places. In other words, we simply say that if door is open then motor component must be stopped otherwise it violates the safety property. These two components don't work at the same time otherwise safety property doesn't hold.

4.4.3 State-Space Query function

For performing the verification of properties, custom functions are built using CPN_ML. Below are some functions.

4.4.3.1 Goal State

Finding goal-state depends on the way in which goal state is defined and finding goal state reachability is not a standard function in CPN. Library functions are used to specify a new predicate and that predicate is then used to search nodes which satisfy that predicate using PredAllNodes () function. Presence of one or two nodes satisfying the predicate means that the goal-state is reachable. PredAllNodes () function is used with the predicate if it is also important to know that how the goal is reachable and what transitions between states will lead to the goal. **Figure 21** illustrate the working of goal reachability algorithm in which it searches all the nodes in graph. We store tokens in variables. We check the desired goal. If the condition is correct it returns true, otherwise, it returns false.

Chapter 5

Composability Verification of an Elevator Model

In this section, we provided the details of the case study of an elevator model we assume an elevator which servers 6 floors. For the sake of simplicity and reduced space, we initialize each florr with 3-4 passengers (though the actual model is capable of taking a large number of incoming passengers).

We discussed composability verification and propose the use of Colored Petri Nets for component oriented model development, composition and the verification of composed models using state-space analysis technique. We presented a case study of an elevator model as a proof of concept. Our case study explains the proposed process of developing and composing CPN based model components, and verifying the composed model using state-space analysis.

Following steps demostrate the composability verification process of our model:

5.1 Requirement Specification

We define RS= $\langle \mathbf{O_1}, \mathbf{S_1} \rangle$ where

Objective $\mathbf{O_1}$ = {All the arrived passengers must reach their destination floor}

Constraint $\mathbf{S_1}$ = {The door should never be opened when the elevator is moving}

5.2 Model Components

Six basic components were developed for the construction of an elevator model:

5.2.1 Generator

Figure 11 illustrate the generator component. It can generate the passenger id, current floor and desired floor. A transition name generator in which we used NextPassenger () function and a guard condition [id<Max] which check the limit of passengers. **Table 3** illustrate the detail of colset that is used in making of motor component.

Table 3: Generator Component Code

Declaration	Explanation
colset tint = INT timed;	Timed colset tint can be used.
colset Passenger = product INT*INT*INT timed;	Define the type used for the Passenger.
colset INT=int;	Integer type colset INT is used.
[id<Max]	Condition is used to check the number of passenger is generated must not exceed more than 25.

if cf<>df then b+1
else b
	We count the total number of passenger that are generated using NextPassenger () function. If current floor is not equal to desired floor then we add plus 1 in variable b. Starting value of b is 0.

fun exponentialTime(mean : int)= let val Mean = Real.fromInt mean val rv = exponential((1.0)/Mean) in floor(rv+0.5) end;	Exponential distribution function is used with a mean equal to 5.

fun NextPassenger(id) =
let
val nb = discrete(1,6);
val nf = discrete(1,6);
in
(id,nf,nb) end;
	Function NextPassenger () is used to generate passenger id, current floor and desired floor range between 1 and 6.

if cf<>df andalso cf=1
then 1`(i,cf,df)
else empty
	We have six floors in our model and using if condition we check if current floor is not equal to desired floor and current floor must be equal to 1 if condition is true then passenger arrived at floor 1.

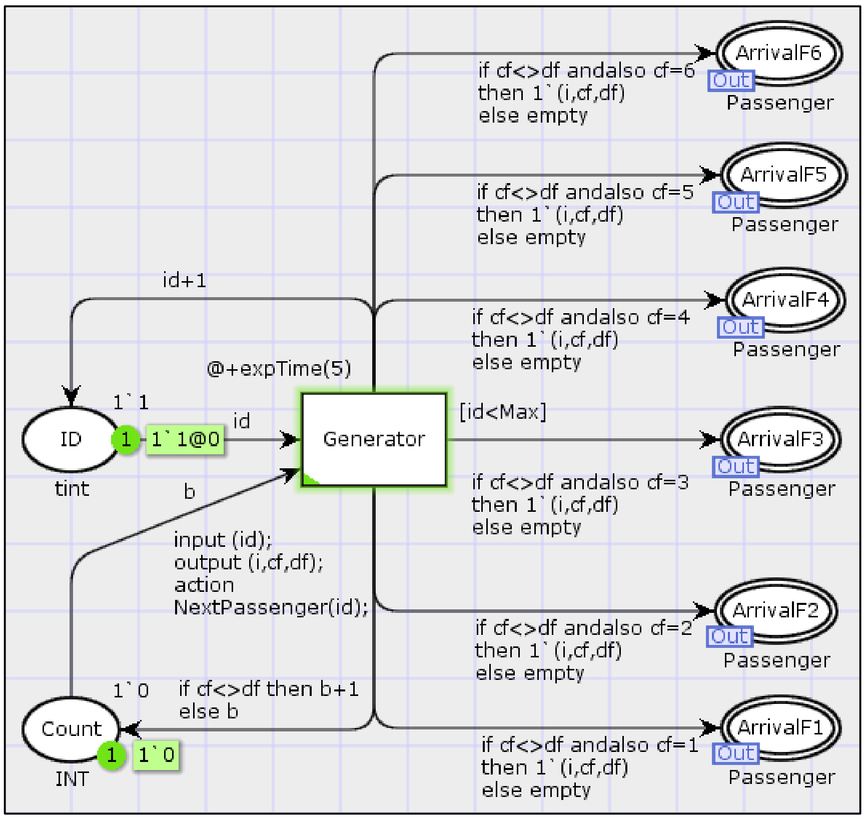


Figure 11: Generator Component

5.2.2 Queue

FIFO queue component is used in order to entertain the passengers. A passenger arriving first will always be served first. **Figure 12** illustrate Queue component. **Table 4** illustrate the detail of colset that is used in making of queue component.

Declaration	Explanation
colset Passenger = product INT*INT*INT timed;	Define the type used for the Passenger.
var i,cf,df:INT;	Integer type variable is used to save integer value that is passenger I represent passenger id, cf represent current floor and df represent desired floor
var p:Passengers;	Variable p is used for Passengers

Table 4: Queue Component Code

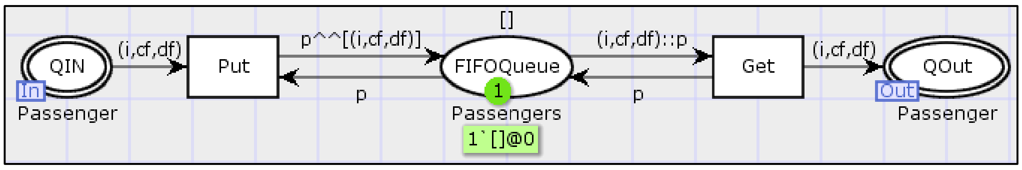


Figure 12: Queue Component

5.2.3 Panel

It is the button panel that is installed on each floor outside the elevator door. When the passengers arrive, they press the panel buttons to call the elevator at their current floor. **Figure 13** illustrate the design of the panel component. It takes the passenger tokens as input in their respective floors, process them in a FIFO queue, construct a list of trips and pass on the passenger tokens to the output. **Table 5** illustrate the detail of colset that is used in making of panel component.

Declaration	Explanation
colset TRIPS = list INT timed;	Timed list type colset TRIPS can be used.
colset Passenger = product INT*INT*INT timed;	Define the type used for the Passenger.
var ps: Passenger;	Variable ps is used for Passenger
var trips: TRIPS;	Variable trips is used for TRIPS
#2 ps::trips	Pick the second element in the ps.
List.nth(trips, 0)	Selects an element from the list trips

Table 5: Panel Component Code

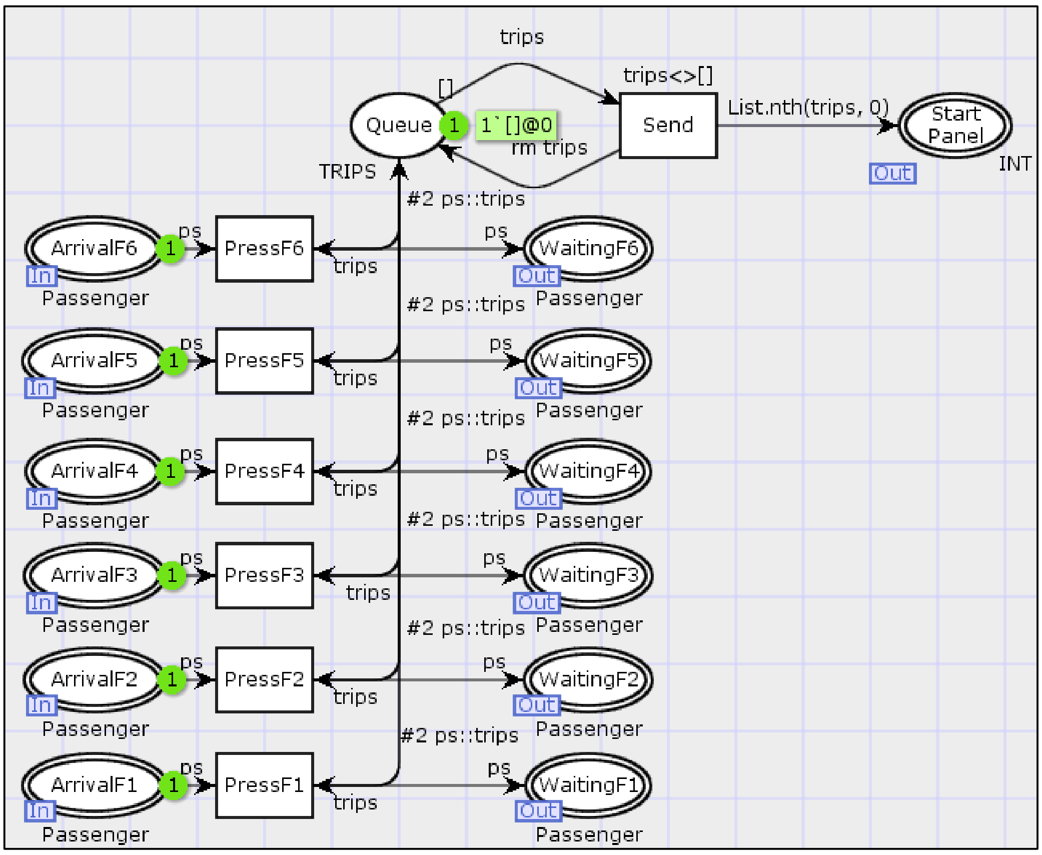


Figure 13: Panel Component

5.2.4 Door

The door opens and closes on arriving at each floor and allows the passengers to enter the cabin according to the capacity. The place 'current floor' represents the current floor. The place 'Load' represents the current load in the cabin. Only those passengers can enter which are at the current floor. The door closes when there are no more passengers or the maximum capacity has been reached. **Figure 14** illustrate the door component. **Table 6** illustrate the detail of colset that is used in making of door component.

Declaration	Explanation
var floor,load:INT;	Integer variables floor and load is used.
colset Passenger = product INT*INT*INT timed;	Define the type used for the Passenger.
var ps: Passenger;	Variable ps is used for Passenger
val MAX_CAPACITY = 10;	MAX_Capacity variable is used to check the number of passenger enter in elevator must not exceeds more than 10.
colset Passengers = list Passenger timed;	Timed list type colset Passengers can be used.
var p:Passengers;	Variable p is used for Passengers
ps::p	FIFO Queue is used in which a passenger is arriving first will be served first.
p<>[]	Timed list type variable p is used and check the condition if p is not equal to empty then value of p is forward to the next place.

Table 6: Door Component Code

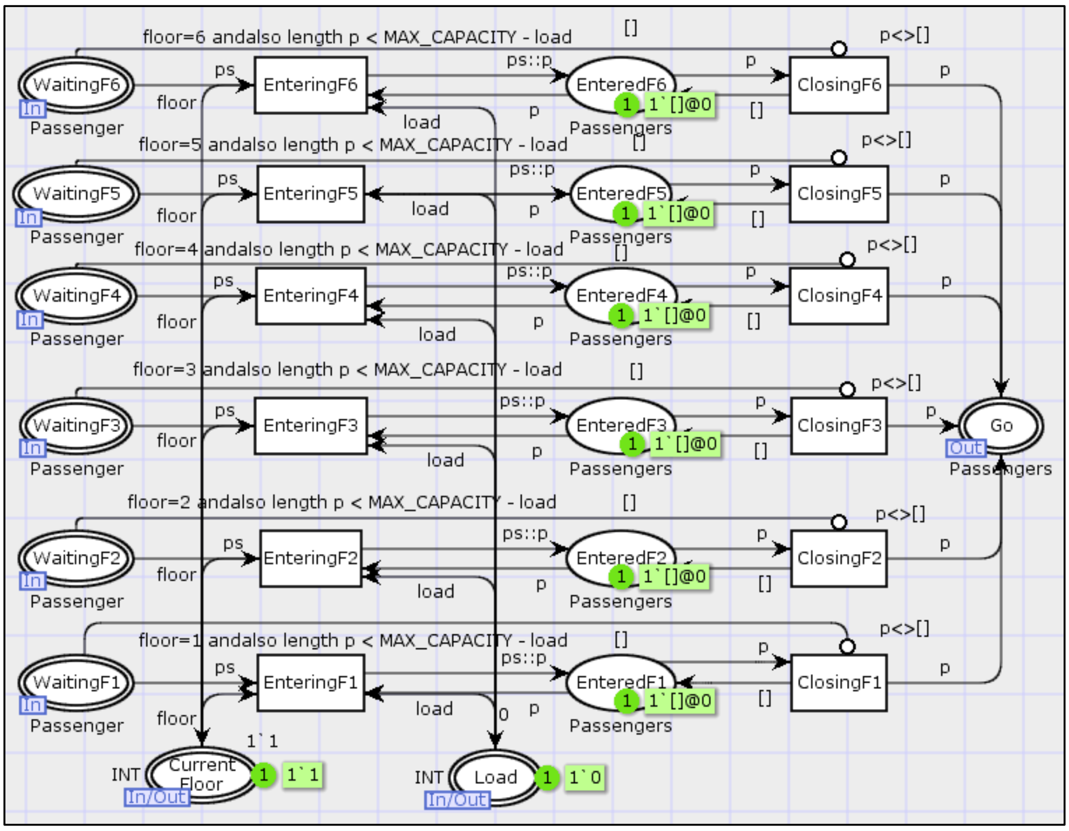


Figure 14: Door Component

5.2.5 Cabin

It is the carriage for a maximum of 10 passengers. When the passengers enter the cabin they wait until their desired floor has arrived. When the passengers enter the cabin, their desired floor is selected and a list of trips is created after removing the duplicates. The passengers wait until the elevator arrives at their desired destination **Figure 15** illustrate cabin component. **Table 7** illustrate the detail of colset that is used in making of cabin component.

Table 7: Cabin Component Code

Declaration	Explanation
colset TRIPS = list INT timed;	Timed list type colset TRIPS can be used.
colset Passenger = product INT*INT*INT timed;	Define the type colset used for the Passenger.
var ps: Passenger;	Variable ps is used for Passenger
var trips: TRIPS;	Variable trips is used for TRIPS
#3 ps::trips	Pick the third element in the ps.
var floor,load:INT;	Integer variables floor and load is used.
trips <>[]	If the value of trips not equal to empty it can generates more trips.
colset INT=int;	Integer type colset INT is used.
[#3 ps = 1 andalso floor = 1]	Condition is used to check the value of current floor and desired floor are equal then door will open and passenger will leave the floor.

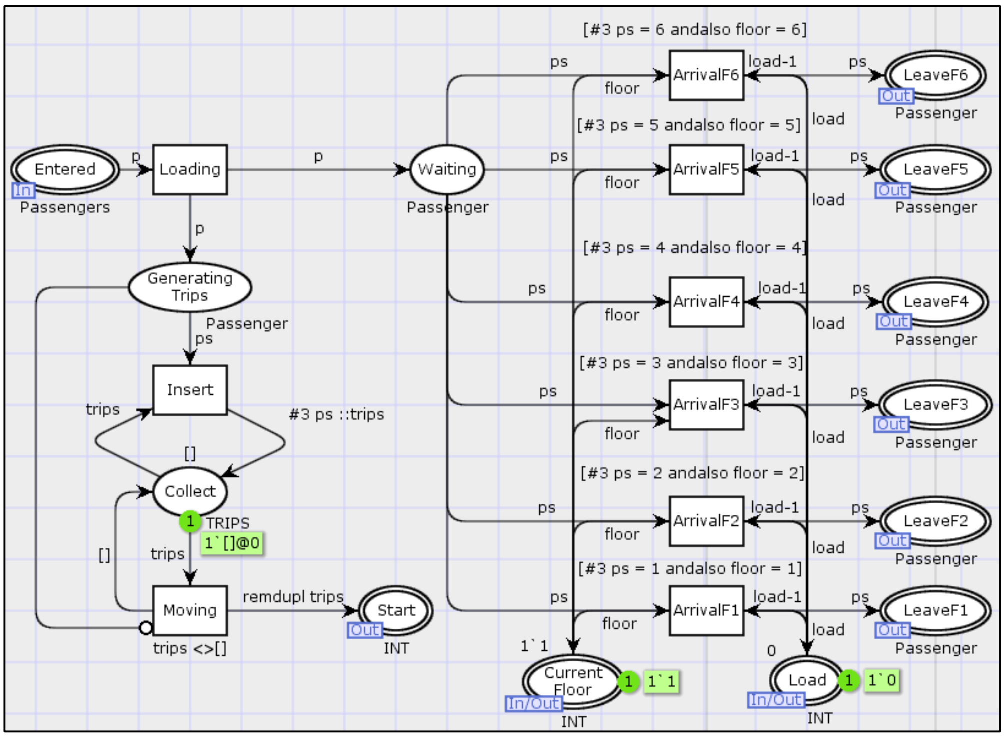


Figure 15: Cabin Component

5.2.6 Motor

It takes a list of trips as input from the cabin (the internal button panel selection for the desired floor) or from the floors (outer panel) and moves the motor right to go upwards or left to go downwards. Motor has brakes, which are applied when the desired floor is reached. This component is responsible to change the 'current floor'. **Figure 16** illustrate Motor component. **Table 8** illustrate the detail of colset that is used in making of motor component.

Declaration	Explanation
colset TRIPS = list INT timed;	Timed list type colset TRIPS can be used.
colset Passenger = product INT*INT*INT timed;	Define the type used for the Passenger.
var ps: Passenger;	Variable ps is used for Passenger
var trips: TRIPS;	Variable trips is used for TRIPS
var floor:INT;	Variable floor is used for Integer.
[trip < floor]	If the value of trips is less than floor then motor will move downward.
[trip > floor]	If the value of trips is greater than floor then motor will move upward.
colset INT=int;	Integer type colset INT is used.
colset MOTOR = product INT*INT timed;	Define the type colset used for the Passenger.
[trip = floor]	If the value of trip is equal to the floor than motor will stop at desired floor.

Table 8: Motor Component Code

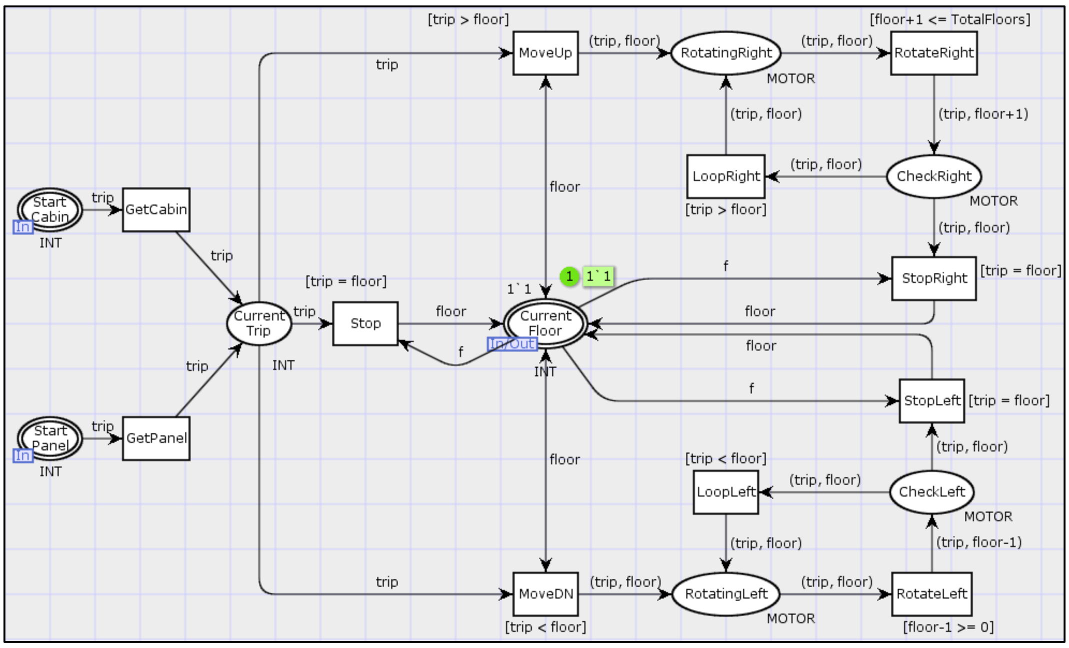


Figure 16: Motor Component

5.3 Model Composition

All the components are composed together to form an elevator model as shown in **Figure 17**.

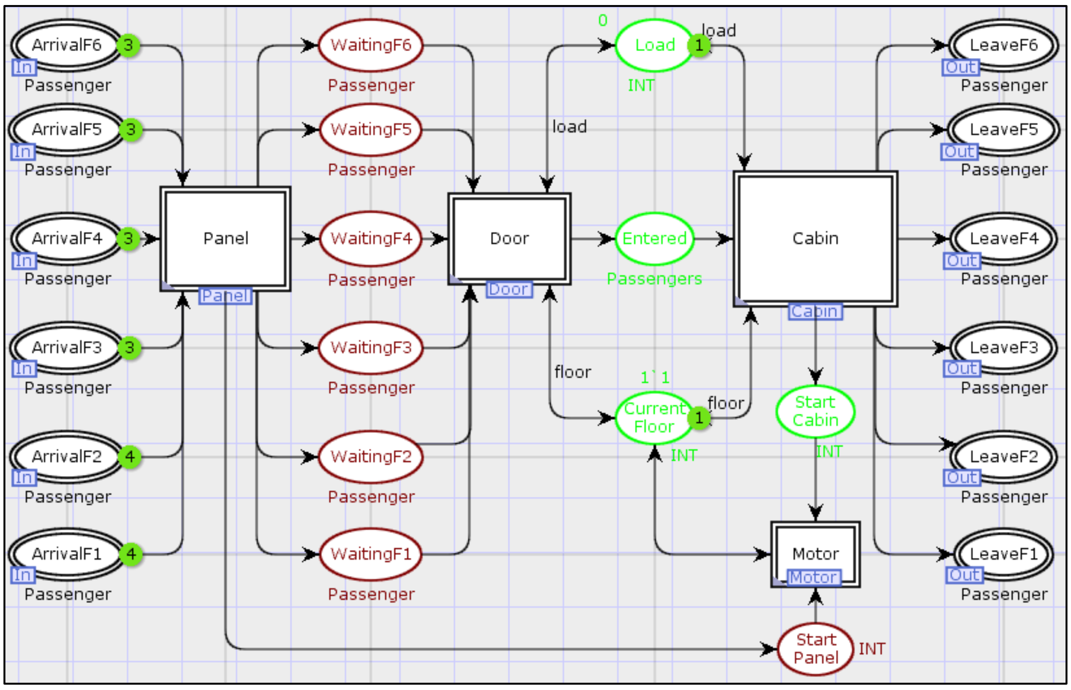


Figure 17: Elevator Composed Model

5.4 Model Execution

Figure 18 & **Figure 19** illustrate the initial state and the final state of the model's execution. The tokens in the initial state represent passengers as a tuple: **{Passenger ID, Current Floor, and Desired Floor}.** Note that in the final state, all the passengers reach their desired floor. **Table 9** illustrate the detail of colset that is used in making of main component.

Table 9: Main Component Code

Declaration	Explanation
colset Passenger = product INT*INT*INT timed;	Define the timed type used for the Passenger.

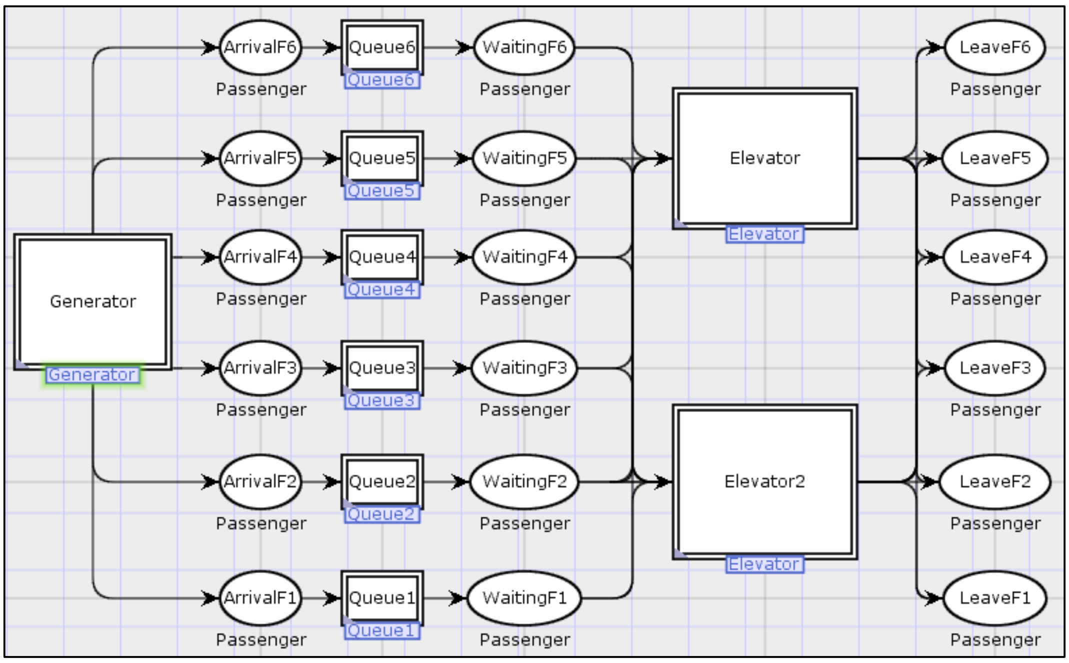


Figure 18: Initial Step

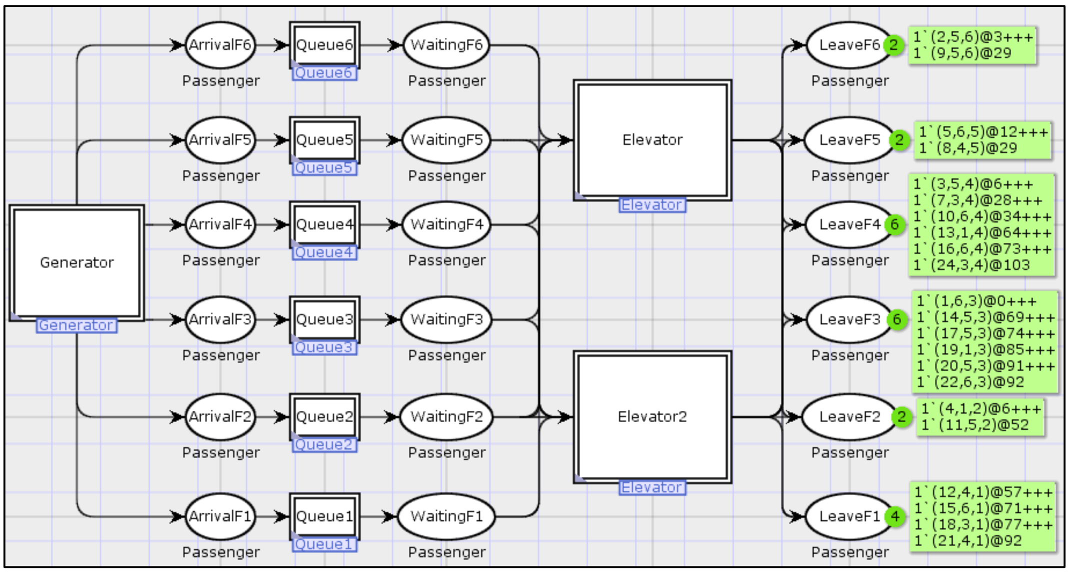


Figure 19: Final Step

5.5 Composability Verification

In this step, we perform the state- space analysis. After generating the state-space of the composed model, we visualize it in Gephi tool as shown in **Figure 20**. In the state space, Node 1 is the initial node and Node 1227 is the goal state. The shortest path to reach the goal state is shown in red color.

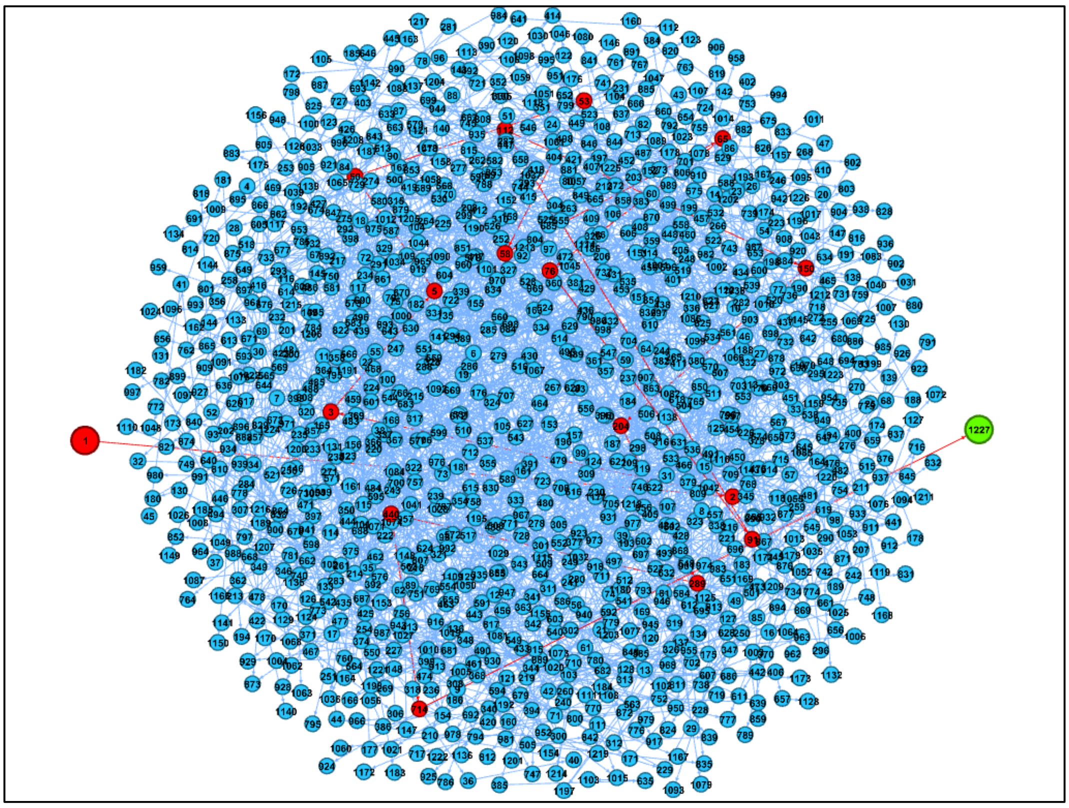
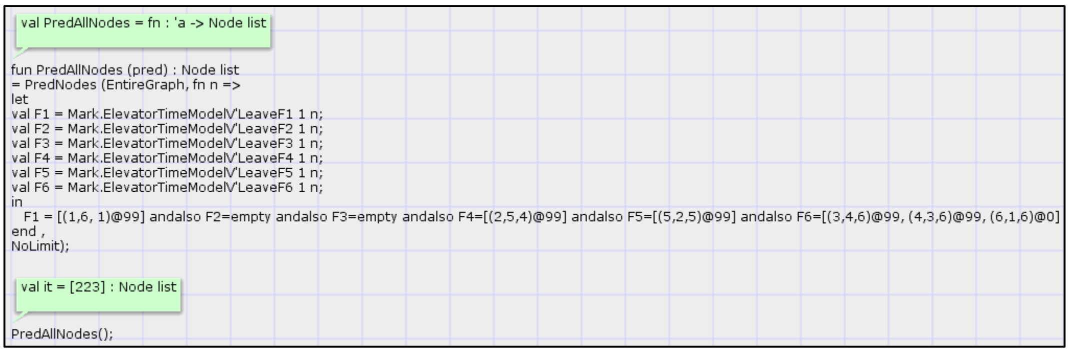


Figure 20: State-Space of Elevator Model

We developed and performed the query functions, shown in Figure to prove that goal state is reachable and constraint will never be reached. The goal is to ensure that all the passengers arrive at their desired floors, so we check that there exists a marking that satisfies this criterion. The constraint is to ensure that the door will never be opened when the elevator is moving. We prove that if there are tokens in 'Entered' place of any floor, meaning the door is opened, then the 'Rotating Left' or 'Rotating Right' place is empty and vice versa. The satisfaction of goals and constraints assert that all the components are consistent and their behavioral composability is verified as per given requirement specification.

Figure 21: Goal State Reachability Query

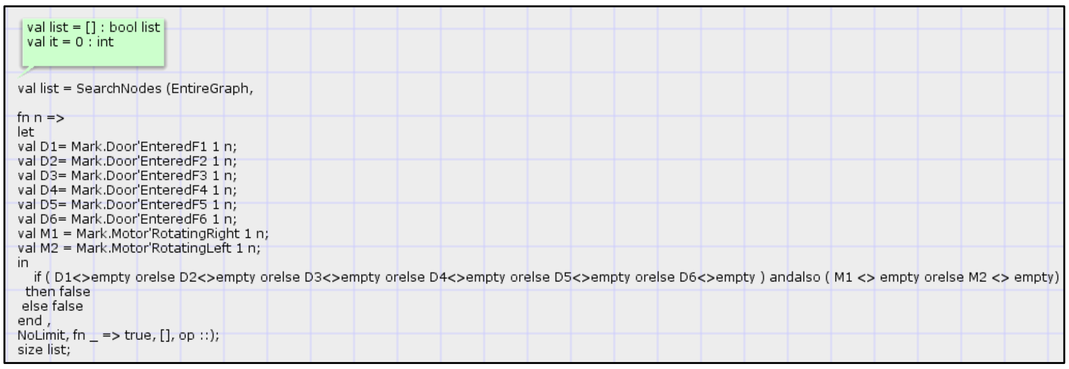


```
val PredAllNodes = fn : 'a -> Node list

fun PredAllNodes (pred) : Node list
= PredNodes (EntireGraph, fn n =>
let
val F1 = Mark.ElevatorTimeModelV'LeaveF1 1 n;
val F2 = Mark.ElevatorTimeModelV'LeaveF2 1 n;
val F3 = Mark.ElevatorTimeModelV'LeaveF3 1 n;
val F4 = Mark.ElevatorTimeModelV'LeaveF4 1 n;
val F5 = Mark.ElevatorTimeModelV'LeaveF5 1 n;
val F6 = Mark.ElevatorTimeModelV'LeaveF6 1 n;
in
  F1 = [(1,6, 1)@99] andalso F2=empty andalso F3=empty andalso F4=[(2,5,4)@99] andalso F5=[(5,2,5)@99] andalso F6=[(3,4,6)@99, (4,3,6)@99, (6,1,6)@0]
end ,
NoLimit);

val it = [223] : Node list

PredAllNodes();
```



```
val list = [] : bool list
val it = 0 : int
```

```
val list = SearchNodes (EntireGraph,

fn n =>
let
val D1= Mark.Door'EnteredF1 1 n;
val D2= Mark.Door'EnteredF2 1 n;
val D3= Mark.Door'EnteredF3 1 n;
val D4= Mark.Door'EnteredF4 1 n;
val D5= Mark.Door'EnteredF5 1 n;
val D6= Mark.Door'EnteredF6 1 n;
val M1 = Mark.Motor'RotatingRight 1 n;
val M2 = Mark.Motor'RotatingLeft 1 n;
in
    if ( D1<>empty orelse D2<>empty orelse D3<>empty orelse D4<>empty orelse D5<>empty orelse D6<>empty ) andalso ( M1 <> empty orelse M2 <> empty)
  then false
 else false
end ,
NoLimit, fn _ => true, [], op ::);
size list;
```

Figure 22: Constraint Unreachability Query

Chapter 6

Conclusion and Future Work

This chapter provides the discussion, conclusion and future work of the thesis.

6.1 Summary

In this thesis, we discussed composability verification and propose the use of Colored Petri Nets for component oriented model development, composition and the verification of composed models using state-space analysis technique. We presented a case study of an elevator model as a proof of concept. Our case study explains the proposed process of developing and composing CPN based model components, and verifying the composed model using state-space analysis.

6.2 Conclusion

A verified composed model ensures consistent structure and compatible behavior of the composites to guarantee the satisfaction of its objectives and required constraints, given in the requirement specifications. This helps in rectifying any possible defects in the model design of a complex system before it is actually implemented to serve its purpose, and thus saves a significant amount of time, cost and achieve robustness. Moreover this process supports reusability as the entire process can easily be repeated to compose same components for different scenarios with varied configurations or with different requirement specifications.

We used verification techniques named as state space analyses. Using state space analysis, we verified the Composability of components and thus ensure successful reusability. We verified different properties in our elevator model including safety, Goal-reachability property. In order to ensure that our model works correctly, we verified these properties. For efficient and accurate verification of models, colored Petri nets and its techniques are very beneficial as it is one of the competitive tools for development of models for systems. CPN's role in Composability verification is very constructive, because it also focuses on dynamic Composability level. CPN communities have been working on the analysis techniques to improve the performance of CPN tools and their work provides a notable and efficient reasoning when working on the model correctness.

6.3 Future Work

In future, we intend to deploy the composability verification framework in different application areas, particularly in safety critical systems, to evaluate its potential and to make use of its valuable features in the verification of complex system design and its correctness analysis. We further aim to extend our approach for heterogeneous model composability.

References

[1] M. Serrano Zanetti, A complex systems approach to software engineering, vol. 2, Germany: Suedwestdeutscher Verlag fuer Hochschulschriften, 2013, p. 184.

[2] I. Mahmood, "A Verification Framework for Component Based Modeling and Simulation: Putting the pieces together," KTH, Stockholm, 2013.

[3] C. Szyperski, Component Software: Beyond Object-Oriented Programming, 2nd ed., Boston, MA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[4] R. G. Bartholet, D. C. Brogan, P. F. Reynolds and J. C. Carnahan, "J.C.: In search of the philosopher's stone: Simulation composability versus component-based software design," in *Proceedings of the Fall Simulation Interoperability Workshop*, Arlington, Virginia, 2004.

[5] M. Mitchell, Complexity: A Guided Tour, New York, NY, USA: Oxford University Press, Inc., 2009.

[6] D. Rickles, P. Hawe and A. Shiell, "A simple guide to chaos and complexity," *Journal of Epidemiology & Community Health,* vol. 61, no. 21, pp. 933-937, 2007.

[7] N. R. Jennings, "An Agent-based Approach for Building Complex Software Systems," *Commun. ACM,* vol. 44, no. 4, pp. 35-41, 4 2001.

[8] B. Meyer, Object-Oriented Software Construction, 1st ed., Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1988.

[9] H. S. Sarjoughian, "MODEL COMPOSABILITY," in *Proceedings of the 2006 Winter Simulation Conference*, Berlin, 2006.

[10] I. Mahmood, R. A. V. V. and F. M. , "Verifying Dynamic Semantic Composability of BOM-Based ComposedModels Using Colored Petri Nets," in *PADS '12 Proceedings of the 2012 ACM/IEEE/SCS 26th Workshop on Principles of Advanced and Distributed Simulation*, Washington, 2012.

[11] M. D. Petty and E. W. Weisel , "A theory of simulation composability," in *Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando, 2004.

[12] S. Kasputis, "Composable Simulations," in *2000 Winter Simulation Conference Proceedings*, Washington, DC, 2000.

[13] P. K. Davis and R. H. Anderson, "Improving the composability of department of defense models and simulations," RAND Corporation, United States, 2003.

[14] F. Moradi, "A Framework for Component Based Modelling and Simulation using BOMs and Semantic Web Technology," KTH, Stockholm, 2008.

[15] O. Balci, "Verification, Validation and Accreditation of simulation models," in *Proceedings of the Winter Simulation Conference*, Atlanta, GA, 1997.

[16] M. D. Petty, "Verification and Validation," in *Principles of Modeling and Simulation: A multidisciplinary approach*, Canada, Wiley; 1 edition, 2009, p. ch. 6.

[17] K. Jensen and L. M. Kristensen, "Colored Petri Nets: A Graphical Language for Formal Modeling and Validation of Concurrent Systems," *Communications of the ACM,* vol. 58, no. 6, pp. 61-70, June, 2015.

[18] I. Mahmood, R. A. V. V. and F. M. , "Composability Verification of Real Time System Models using Colored Petri Nets," in *UKSim 15th International Conference on Computer Modelling and Simulation*, Washington, DC, 2013.

[19] I. Mahmood, "A Verification Framework for Component Based Modeling and Simulation : Putting the pieces together," KTH, Stockholm, 2013.

[20] M. D. Petty, "Verification and Validation," in *Principles of Modeling and Simulation*, New York, John Wiley & Sons, 2009, pp. 121-149.

[21] S. Robinson, R. Brooks, K. Kotiadis and D. J. Van Der Zee, Conceptual Modeling for Discrete-Event Simulation, 1st ed., Boca Raton, FL, USA: CRC Press, Inc., 2010.

[22] P. A. Fishwick, Simulation Model Design and Execution: Building Digital Worlds, 1st ed., Upper Saddle River, NJ, USA: Prentice Hall PTR, 1995.

[23] K. Jensen and L. M. Kristensen, Coloured Petri Nets Modelling and Validation of Concurrent Systems, New York: Springer-Verlag Berlin Heidelberg, 2009, p. 382.

[24] C. Tools, "CPN Tools," December 2017. [Online]. Available: http://cpntools.org/.

[25] M. Naedele and J. W. Janneck, "Design patterns in petri net system modeling," in *Fourth IEEE International Conference on Engineering of Complex Computer Systems*, Monterey, 1998.

[26] L. M. Kristensen, "State Space Methods for Coloured Petri Nets," Kluwer Academic, Denmark, 2000.

[27] K. Jensen, S. C. and L. M. Kristensen, "CPN Tools State Space Manual," Department of Computer Science, Aarhus, 2006.

[28] K. Jensen, S. Christense and L. M. Kristensen, "CPN Tools State Space Manual," Department of Computer Science, Aarhus , Denmark, 2006.

[29] K. K. Lau and Z. Wang, "Software Component Models," *IEEE Transactions on Software Engineering,* vol. 33, no. 10, pp. 709-724, 10 2007.

[30] M. R. J. Qureshi and S. A. Hussain, "A reusable software component-based development process model," *Advances in Engineering Software,* vol. 39, no. 2, pp. 88-94, 2008.

[31] N. M. J. Basha and S. A. Moiz, "Component based software development: A state of art," in *IEEE-International Conference On Advances In Engineering, Science And Management*, Singapore, 2012.

[32] E. Okewu and O. Daramola, "Component-based software engineering approach to development of a university e-administration system," in *2014 IEEE 6th International Conference on Adaptive Science Technology (ICAST)*, Covenant University, Canaanland, Ota, Ogun State,Nigeria., 2014.

[33] O. Balci, J. D. Arthur and W. F. Ormsby, "Achieving reusability and composability with a simulation conceptual model," *Journal of Simulation,* vol. 5, no. 3, pp. 157-165, 01 8 2011.

[34] M. D. Petty and E. W. Weisel, "A Composability Lexicon," in *in Proceedings of the Spring 2003 Simulation Interoperability Workshop*, Orlando, April 2003..

[35] A. Engel, Verification validation and testing of engineered systems, Hoboken, NJ: J Wiley & Sons, 2010.

[36] B. Meyer, Object-oriented Software Construction (2nd Ed.), Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1997.

[37] M. Debbabi, F. Hassane, Y. Jarraya, A. Soeanu and L. Alawneh, Verification and Validation in Systems Engineering: Assessing UML/SysML Design Models, 1st ed., New York, NY, USA: Springer-Verlag New York, Inc., 2010.

[38] M. K. Ganai and A. Gupta, SAT-Based Scalable Formal Verification Solutions, USA: Springer, 2007.

[39] C. A. Petri, Kommunikation mit Automaten, University in Darmstadt, Germany: Bonn : Mathematisches Institut der Universität Bonn, 1962.

[40] W. J. Thong and M. Ameedeen, "A Survey of Petri Net Tools," in *Proceedings of the 1st International Conference on Communication and Computer Engineering*, Malacca, 2015.

[41] M. Naedele and J. W. Janneck, "Design patterns in Petri net system modeling," in *Proceedings. Fourth IEEE International Conference on Engineering of Complex Computer Systems*, Monterey, California, 1998.

[42] L. G. Xue and Y. W. Yao, "A Method for Modeling Multi-task Workflow Based on High-Level Petri Nets," in *2009 Second International Workshop on Computer Science and Engineering*, Canada, 2009.

[43] N. Mulyar and W. M. v. d. Aalst, "Towards a Pattern Language for Colored Petri Nets," in *Conference: Proceedings of the Sixth Workshop on the Practical Use of Coloured Petri Nets and CPN Tools (CPN 2005)*, Netherland, 2005.

[44] W. Reisig, Understanding Petri Nets: Modeling Techniques, Analysis Methods, Case Studies, Berlin, Germany: Springer Publishing Company, Incorporated, 2013.

[45] W. M. P. Aalst, C. Stahl and M. Westergaard, "Strategies for Modeling Complex Processes Using Colored Petri Nets," in *Transactions on Petri Nets and Other Models of Concurrency VII*, Berlin, 2013.

[46] S. McConnell, Code Complete, Second Edition, Redmond, WA: Microsoft Press, 2004.

[47] Matthew Wilson, "Quality Matters: Correctness, Robustness and Reliability," *Overload Journal Process Topics,* vol. 11, no. 93, pp. 23-34, October 2009.

[48] L. D. d. Silva and A. P. , "A Model-Based Approach to Formal Specification and Verification of Embedded Systems Using Colored Petri Nets," in *Component-Based Software Development for Embedded Systems*, Berlin, Heidelberg, Springer-Verlag, 2005., pp. 35--58.

[49] K. C. Gorgonio and A. P. , "Adaptation of Coloured Petri Nets Models of Software Artifacts for Reuse," in *ICSR-7 Proceedings of the 7th International Conference on Software Reuse: Methods, Techniques, and Tools*, London, 2002.