*Proceedings of the 2021 Winter Simulation Conference*
*S. Kim, B. Feng, K. Smith, S. Masoud, Z. Zheng, C. Szabo and M. Loper, eds.*

# COMPOSABILITY VERIFICATION OF COMPLEX SYSTEMS USING COLORED PETRI NETS

Imran Mahmood
Syed Hassan Askari

Center for Research in Modeling & Simulation
School of Electrical Engineering and Computer Science
National University of Sciences and Technology
H-12 Sector, Islamabad, Pakistan

Hessam S. Sarjoughian

Arizona Center for Integrative Modeling & Simulation
School of Computing, Information, and Decision Systems Engineering
Arizona State University, Tempe, Arizona, USA

## ABSTRACT

The discipline of component-based modeling and simulation offers promising gains including reductions in development cost, time, and system complexity. This paradigm 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. In this paper, 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.

## 1 INTRODUCTION

The Modeling & Simulation(M&S) community continues research on the path for novel methods that can reduce development cost and time through managing system complexity. M&S development processes are inherently complicated and resource-intensive. Complex models require a great deal of time and effort to develop and evaluate. Researchers and practitioners continue exploring theoretical fundamentals, quality design principles, and supportive methods, techniques, and tools to respond to these challenges. The practice of model reuse for developing simulation models, supported with advances in software engineering principles, technologies, and practices, has been successful. However, more research is needed, especially as multifaceted systems continue to become larger and more complex. The extensive use of reusable models offers numerous advantages, including reductions in development cost, time, and system complexity, and allows logical partitioning of complex systems into parts. There is an increasing trend to build simulation models by reusing and repurposing existing models in support of model composability (Balci et al. 2017; Davis 2006; Kasputis and Ng 2000).

Inspired by the discipline of Component-based Software Engineering, models are built using "Model Components, commonly referred to as building blocks. A model component is an independent element defined using a concise modeling language with a well-defined interface and encapsulated behavior. A model component can be independently deployed subject to third-party composition with or without modification (e.g., (Dahmann et al. 1997; Mahmood 2013)). Model components can be composed if their interfaces match each other using formal and semi-formal modeling languages. However, it can be difficult

**978-1-6654-3311-2/21/$31.00 ©2021 IEEE**

to claim models are correctly composed to fulfill desired functional requirements without verification and validation.

Model Composability is a crucial design characteristic and an essential means toward achieving reusability. Yet it is a known daunting challenge in the M&S community. System theory can be used to define discrete time conjunctive, cascade, pure feedback, and mixed to composite model types (Wymore 1993). Other well-known modeling formalisms supporting composition are Automata Theory (Alur 2015; De Alfaro and Henzinger 2001), DEVS (Zeigler et al. 2000), and Petri nets (Petri and Wolfgang 2008). For system-of-systems, it is necessary to support the composition of different kinds of models. Model composition for mixed continuous and discrete models include Hybrid Automata (Henzinger 2000) and Differential Dynamic Logic (Platzer 2008). For composing other kinds of modeling methods (e.g., Linear Programming and Composable Cellular Automata the Knowledge Interchange Broker has been developed (Mayer and Sarjoughian 2009; Sarjoughian 2006). The Linear Temporal Logic (LTL), Computation Tree Logic (CTL), Metric Temporal Logic (MTL), and others may be used to define requirements. Such formally specified system requirements can be verified for dynamical models that are developed using different monolithic or heterogeneous modeling methods.

Model composability from technology-oriented vantage points can be viewed from syntactic and semantic compatibility aspects (Dahmann et al. 1997; Mahmood et al. 2012; Moradi et al. 2007; Tolk 2010; Zhu et al. 2018). Syntax focuses on the composition of models according to their given modeling language(s). Syntactic composability for monolithic formal modeling methods such as Petri net, unlike semantic composability, is simple. Semantic composability aims to satisfy the expected behavior of the whole given the expected behaviors of the parts and their relationships. A common way to satisfy behavioral composability is to examine and show all possible states and their transitions are satisfied for a given requirement specification. Behavioral composability among components can only be achieved if the components are at the right states during their interaction and they possess the required behavior to make mutual progress. It is also important to structure and behavior for model composition. Structural composition is simpler as it is focusing on the connections in flat and hierarchical models. Behavioral composition, in contrast, is not simple as it must account for encapsulated behaviors of individual and composed model components.

### 1.1 Model components and composability

Model composability is fundamental to modeling and simulation. Compositions of models possess precise structural and behavioral properties in order to fulfill a given requirement specification. The structural and behavioral parts of model composability are concerned with the accuracy of transforming user requirements into models that can be verified and validated. A description and methodology for verification, validation and accreditation with community support provides comprehensive details and nuances for real-world use (Gholami and Sarjoughian 2017; Pace 2004; Petty 2008; Sargent 2005; SISO 2007). Verification deals with model correctness – it is concerned with building the model right. This aims to specify a model that is built according to the constructs of a modeling language and requirements specifications. Verification shows a system's desired structure and behavior are correctly specified. Validation is concerned with building the right model. That is, even though a model is built correctly, it may not be what is needed (the model behaves in ways that are not asked for or expected). Correctness for verification is showing the model satisfies certain general properties such as liveness holds true. Correctness for validation shows a collection of specific behaviors that are observed for individual and composed components of a model.

We previously proposed a requirement specification template to specify objectives and constraints as requirement properties in (Mahmood et al. 2012). Thus, a requirement specification is a set of goals (or objectives) that must be reached by the collective participation of the composed components and property constraints that must be satisfied by the composed model throughout its execution. The goals (or objectives) can be defined as reachable "final state(s)" of the composed components or certain desirable combined output (emergent behavior) which individual components cannot produce. Similarly, the property constraints are the system properties such as deadlock freedom, live-lock freedom, fairness (Mahmood et

al. 2011) mutual exclusion or any other scenario specific property that must be satisfied. The proposed requirement specification template essentially helps the modelers build behavioral specification criteria used to evaluate the behavioral composability of the composition to ensure correct reuse of model components in building a complex engineering design. A model checking approach is also used in the behavioral composability verification (Mahmood et al. 2019).

### 1.2 Modeling and Analysis using Petri Nets

Petri nets (PN) is a prominent modeling formalism for modeling complex systems. It provides a graphical notation and a mathematical abstraction for modeling concurrent systems and their behaviors (Petri and Wolfgang 2008). PN graphs intuitively capture structural and behavioral abstractions of Discrete Event Systems. PN community consolidates a variety of analysis techniques that have been developed for decades and are used for reasoning about structural and behavioral properties of PN models. These techniques include reachability analysis, state-space analysis, and model-checking and ease the task of model verification (David and Alla 2010).

Colored Petri Nets (CPN) is an extension of PN characterized by high-level tokens, i.e., places are marked with structured tokens representing data. CPN is a graphical language for constructing models of concurrent systems and analyzing their properties. CPN is a general-purpose discrete event language that combines the capabilities of PN, as a foundation of the graphical notation and a programming language (CPN ML), which is based on Standard ML functional programming language, that provides the primitives for the definition of data types and for specifying data manipulation routines (Jensen et al. 2006; Jensen and Kristensen 2009).

State-space analysis is one of the prominent approaches for conducting formal verification and analyzing the behavior of a model. The basic idea in this approach is to calculate all possible system states and the events which cause the change of states and represent them in the form of a directed graph. When the graph is completely constructed, different search techniques can be applied to analyze the model. A similar approach proposes algorithms to generate a finite-vertex graph, called a reachability graph, via manually translating DEVS model to Time Automata model k (Hwang and Zeigler 2009).

In this paper, we propose the use of Colored Petri Nets for the component-oriented model development using CPN Tools (CPN Tools 2017). We present the steps to (i) construct the structure and behavior of a component using CPN formalism, (ii) specify its input/output interfaces for integration with other components, and (iii) integrate components to form a composed model. We further discuss our composability verification approach using state-space analysis technique. We present a case study of an elevator model as a proof of concept. Our case study explains the process of developing and composing CPN-based model components, and verifying the composed model using state-space analysis.

The rest of the paper is organized as follows: Section 2 briefly defines and explains basic concepts and formalisms used in this paper. Section 3 formulates our methods and approach for model composition and verification. Section 4 furnishes our case study of an Elevator model to explain our approach and section 5 frames the summary and conclusion.

## 2 DEFINITIONS AND CONCEPTS

### 2.1 Colored Petri Nets

CPN allows to define token using data types and complex data manipulation each token has attached a data value called the token color. The token colors can be investigated and modified by the occurring transitions. “CPN Tools” is a software package CPN Tools (CPN Tools 2017) for editing, simulation, state-space analysis, and performance analysis of CPN models. The tool acts as an integrated development environment (IDE) for the construction of CPN models. The most important feature of CPN tool from our point of view is the generation and analysis of state-spaces. The analysis of state-space includes various built-in state-space querying functions and support for creating analysis report which altogether greatly contributes to

the verification process. Figure 1 illustrates the CPN is formally defined by the tuple (Jensen and Kristensen 2009; Askari et al. 2019).

**CPN = (P, T, A, Σ, V, C, G, E, I)**

**Where:**
**P** is a finite set of places
**T** is a finite set of transitions such that: **P ∩ T = ∅**
**A ⊆ P×T ∪ T×P** is a set of directed arcs.
**Σ** is a finite set of non-empty color sets.
**V** is a finite set of typed variables such that: **Type [v] ∈ Σ** for all variables **v ∈ V**
**C: P→Σ** is a color set function that assigns a color set to each place.
**G: T → Expression** is a guard function that assigns a guard to each transition **t**
**E: A→ Expression** is an arc expression function that assigns an arc expression to each arc **a**
**I: P → Expression** is an initialization function that assigns an initialization expression to each place **p**

Figure 1: Colored petri nets tuple.

### 2.1.1 Hierarchical CPN

CPN model can be organized as a set of modules. Each module can be seen as black boxes which make it possible to work at different abstraction levels, concentrating on one at a time. We consider these modules as Model Components. They can be connected to each other with ports and sockets. A socket connects a component at its upper or lower hierarchical level. A port is a place within the body of a component, marked with one of the port-type tags: (i) In (ii) Out or (iii) In/Out and is bound with a socket in the upper level. This relationship is used to define how a component should be connected with the other components.

## 2.2 State-Space Analysis

State-space analysis is one of the prominent approaches for conducting formal analysis and verification. In contrast to algebraic techniques, it is relatively simpler approach for analyzing the behavior of a model. The basic idea in this approach is to calculate all possible system states in a directed graph where vertices represent the state of a system at a particular instance and the edges represent the transitions that cause the change of state. When the graph is completely constructed, different search techniques can be applied to analyze the model. In PN terms, this method is also commonly known as Reachability graph analysis. The state-space analysis of a PN model is performed by exhaustively generating all the states, also called markings. A marking is the number of tokens on the individual places which together represent the state of the system. When the state-space is generated, then reasoning about the PN properties of the model is done by examining the structure of the reachability graph. A constructed state space can help in answering a large set of analytical questions concerning the structure and behavior of the model such as verifying deadlock freedom, absence of live-locks; presence of liveness, the possibility of being able to reach good states, and impossibility of reaching bad states and the guarantee of fulfilling the objectives. For details, readers are referred to (Jensen et al. 2006).

# 3 COMPOSABILITY VERIFICATION PROCESS

In this section we discuss our proposed process of model components development, model composition and the composability verification using state-space analysis technique. Our proposed verification process consist of the following steps:

### 3.1 Requirement Specification

Figure 2 illustrates the modeler gathers the information and formulates the requirement specifications using our proposed template:

**RS = ⟨O, S⟩**
where: O = {o1, o2, o3 …, on} is a set of objectives
S = {s1, s2, s3 …, sn} is a set of constraints

Figure 2: Requirement specification.

#### 3.1.1 Objectives

In modeling terms, an objective $o_i \in O$ 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.

#### 3.1.2 Constraints

In modeling terms, a system constraint $sj \in S$ 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 constraints 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 that 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.

### 3.2 Development & Composition of Model Components

In this step, modelers develop individual components using CPN Tools. A CPN component includes: (i) places defined by primitive or structured data-types called colors; (ii) tokens representing instances of colors of different places; (iii) transitions which consist of events, actions, and time delays; (iv) arcs with arc variables to transport tokens from one element to another; and communicating ports used to expose the In/Out interfaces of the component. All these are packaged together to form a functional Model Component. These components are composed of other components using communicating ports. Section 4 illustrates an example of components of an Elevator model and their composition.

### 3.3 Verification of the Composed Modeling using State-Space Analysis

In the next step, the Composed CPN model is verified using state-space analysis. At first, the state-space of the model is generated using CPN state-space calculation tool. When the state-space is successfully generated, different query functions can be executed to explore the state-space graph for various verification questions.

A query function is like an algorithm that explores the state-space graph. These algorithms are based on theoretical concepts of Petri Nets state-space analysis and are used to verify PN properties. Therefore we translate a system property given in the requirement specification into a suitable PN property. There have been a lot of contributions in the PN literature in specifying PN properties and methods of reasoning of their satisfaction or violation. Figure 3 illustrates the state-space analysis of a composed CPN model using a query function. While writing a query function (algorithm) a modeler is usually searching for one or many suitable marking(s) in the state-space that represent the goal state of the composed model. Similarly, a query function for a constraint maybe the absence of certain marking(s) in the entire state-space. We

illustrate both cases in our case study, later in this paper. When all the goal states are verified and when all the constraints are fulfilled, we say that the composability of the given set of components is verified at the behavioral level.

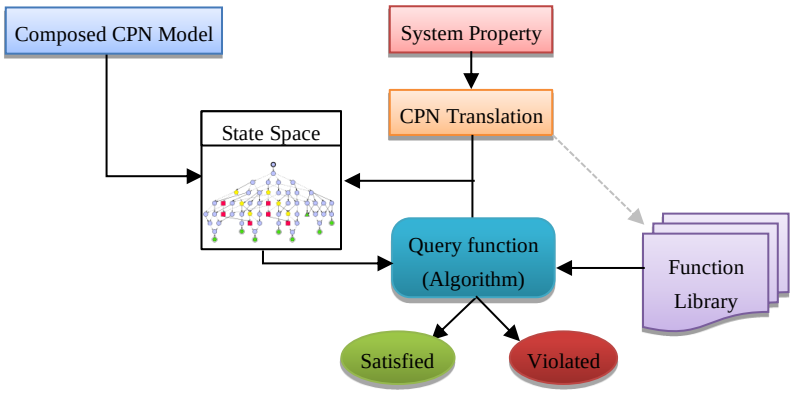



Figure 3: State-Space analysis process.

We can also analyze the state space by means of a Computation Tree Logic. It is possible to formulate queries about states, and the state changes (e.g., the occurrence of certain transitions). Strongly connected components are used to make the model checking for Hierarchical Colored Petri Nets more efficient (Xia et al. 2013).

## 4 CASE STUDY: COMPOSABILITY VERIFICATION OF AN ELEVATOR MODEL

In this section, we provide the details of the case study of an elevator model. We assume the elevator which servers 6 floors. For the sake of simplicity and reduced space, we initialize each floor with 3-4 passengers (though the actual model is capable of taking a large number of incoming passengers). The following steps demonstrate the composability verification process of the elevator model.

### 4.1 Requirement Specification

We define RS = ⟨O1, S1⟩ where, objective O1 = {All the arrived passengers must reach their destination floor} and Constraint S1 = {The door should never be opened when the elevator is moving}.

### 4.2 Model Components

The four basic components developed for the construction of an elevator model are briefly described next.

- 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. It takes the passenger tokens as input in their respective floors, processes them in a FIFO queue, constructs a list of trips and passes on the passenger tokens to the output. System Property State-Space CPN Translation Composed CPN Model Query function (Algorithm) Function Library Satisfied Violated.
- 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.
- 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.

- 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. The motor has brakes, which are applied when the desired floor is reached. This component is responsible to change the 'current floor'. The model files, detailed description and the step-by-step tutorial on how to execute and verify the elevator model is available at: https://github.com/imahmood786/CPN.

### 4.3 Model composition

All the components are composed together to form an elevator model as shown in Figure 4.

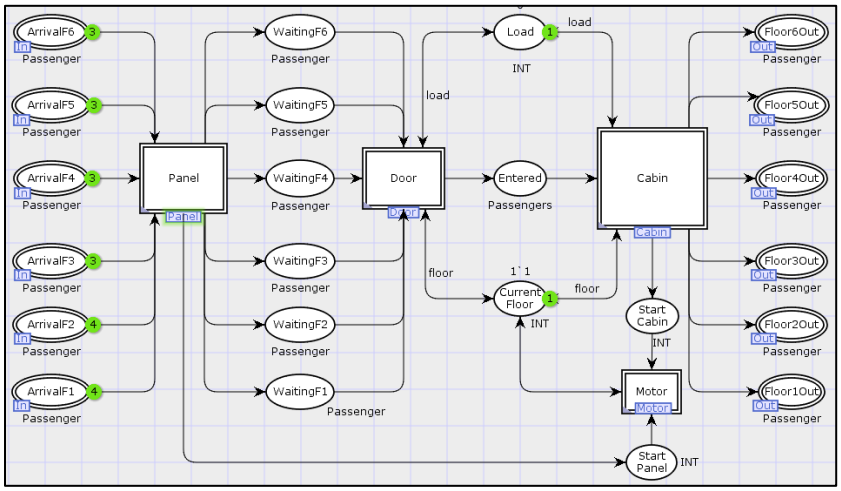
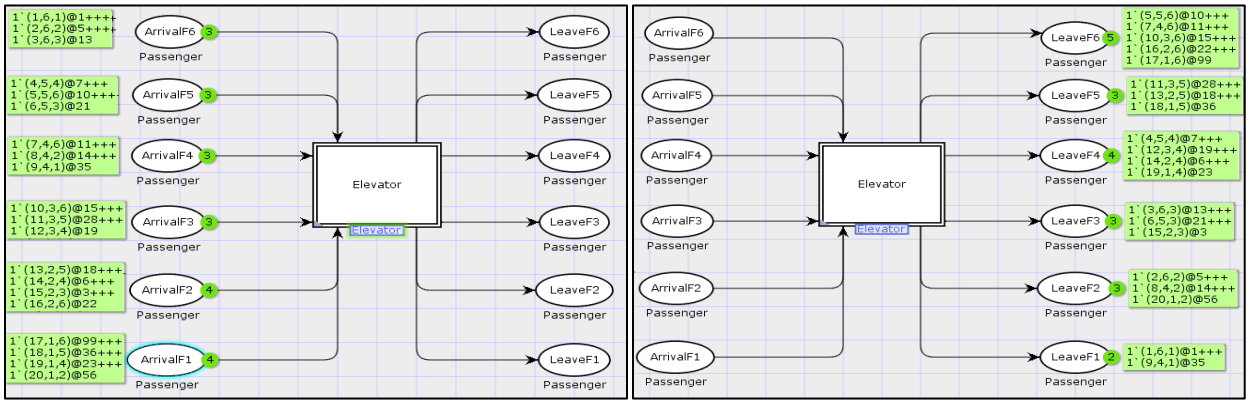



Figure 4: Elevator composed model.

### 4.4 Model Execution

Figure 5 illustrates the initial state and the final state of the model execution. The tokens in the initial state represent passengers as a tuple: **{Passenger ID, Current Floor, Desired Floor}**. Note that in the final state, all the passengers reach their desired floor.




(a) Initial state (b) Final state

Figure 5: State of the model execution.

### 4.5 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 6(a). In the state-space, Node 1 is the initial node and Node 223 is the goal state. The shortest path to reach the goal state is shown in red color. We develop and perform the query functions, shown in Figure 6(b) to prove that the goal state is reachable and ensure that all the passengers arrive at their desired floors. 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 'RotatingLeft' or 'RotatingRight' 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 the given requirement specification.

```
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();
```

(a) Goal state reachability query

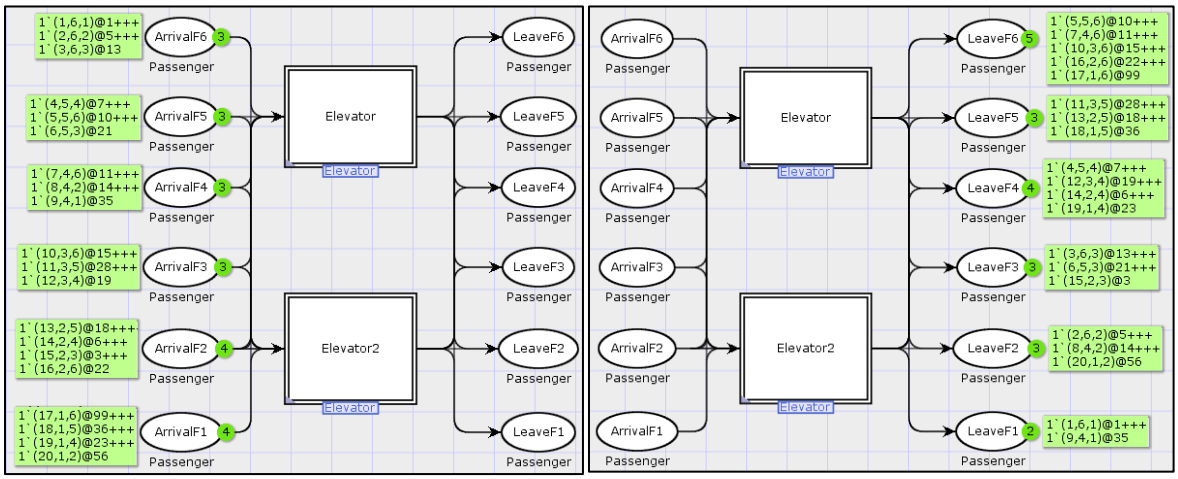
```
val list = [] : bool list
val it = 0 : int

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 true
else 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;
```

(b) Constraint unreachability query.

Figure 6: State-space analysis results.

### 4.6 Reusability of Model Components

We create another scenario of the Elevator Model where two elevators are used to show the reuse of model components. Figure 7(a) shows the initial step of the simulation after reuse and Figure 7(b) show the final step. The reuse of the elevator component renders the same results, it however improves the overall efficiency of the system as the passengers randomly select either elevator and reach their final destination in lesser time. When we apply our composability verification process the goal state is reached and the safety property is satisfied. Thus, we can say that a verified composed model satisfies its requirement specification and that successful composability verification is an important characteristic of model reuse.




(a) Initial step (b) Final step.

Figure 7: Model reuse.

### 4.7 State-Space Reduction Technique

In order to deal with the scalability of the proposed approach and to deal with the state-space explosion we propose a state-space reduction technique. The main idea of this technique is to only consider the places in the composed model and ignore all the places of the underlying components, as we treat them as black boxes. The inputs and outputs of each component can be observed using the flow of Tokens and the data they carry. Therefore in the state-space graph we only keep the markings in which any token is present in the Main model (i.e., any of the place in the main model has at least one token) and delete all other nodes in the state-space graph. The resultant graph will be a reduced form of the actual graph and only considers those markings that reflect a compositional state-space.

## 5 SUMMARY AND CONCLUSION

In this paper, we presented composability verification and proposed the use of Colored Petri Nets for component-oriented model development and the verification of composed models using state-space analysis technique. We presented the proposed process using our case study of an elevator model as a proof of concept. A verified composed model ensures consistent structure and compatible behavior of the composites to guarantee or not the satisfaction of its objectives and required constraints provided in the requirement specifications. Verification helps in rectifying knowable defects in the model design of a complex system before it is actually implemented to serve its purpose and thus helps save a significant amount of time and cost while also achieving design robustness. Moreover, this process supports reusability as the entire process can systematically be repeated to compose existing components for different scenarios with varied configurations or with different requirement specifications. In the future, we intend to deploy the composability verification framework in different application areas, particularly in safety-critical systems, to evaluate its potential and use its valuable features in the verification of complex system design and its correctness analysis. We also aim to extend our approach for heterogeneous model composability.

## ACKNOWLEDGMENTS

Anonymous referees provided constructive reviews of an earlier version of this paper. We thank the referees for their critiques and suggestions that helped improve the presentation of this research.

## REFERENCES


Alur, R. 2015. *Principles of cyber-physical systems*. Cambridge, MA: MIT press.

Askari, Syed H., S. A. Khan, M. Haris, and M. Shoaib. 2019. "Pattern Based Model Reuse Using Colored Petri Nets". In *Proceedings of the 19th International Conference on Computational Science and Its Applications (ICCSA)*, July 1st–4th, St. Petersburg, Russia, 32–38.

Balci, Osman, G. L. Ball, Katherine L. Morse, E. Page, Mikel D. Petty, A. Tolk, and Sandra N. Veautour. 2017. "Model Reuse, Composition, and Adaptation". In *Research Challenges in Modeling and Simulation for Engineering Complex Systems*, edited by R. Fujimoto, C. Bock, W. Chen, E. Page, and J. Panchal. 87–115. Springer, Cham.

CPN Tools. 2017. A tool for editing, simulating, and analyzing Colored Petri nets. http://cpntools.org/, accessed 7th July.

Dahmann, J. S., R. M. Fujimoto, and R. M. Weatherly. 1997. "The department of defense high level architecture". In *Proceedings of the 1997 Winter Simulation Conference*, edited by S. Andradottir, K. J. Healy, D. H. Withers, and B. L. Nelson, 142–149. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

David, R., and H. Alla. 2010. *Discrete, Continuous, and Hybrid Petri Nets*. Berlin: Springer-Verlag Berlin Heidelberg.

Davis, P. K. 2006. *Defense Modeling, Simulation, and Analysis: Meeting the Challenge*. Washington, DC: The National Academies Press.

De Alfaro, L. and T. A. Henzinger. 2001. "Interface automata". *ACM SIGSOFT Software Engineering Notes* 26 (5): 109–120.

Gholami, S. and H. S. Sarjoughian. 2017. "Modeling and verification of network-on-chip using constrained-DEVS". In *Proceedings of the Symposium on Theory of Modeling & Simulation*, April 13rd–16th, Tampa, USA, 1–12.

Henzinger, T. A. 2000. "The Theory of Hybrid Automata". In *Verification of Digital and Hybrid Systems*, edited by M.K. Inan and R. P. Kurshan. 265–292. Springer, Berlin, Heidelberg.

Hwang, M. H., and B. P. Zeigler. 2009. "Reachability Graph of Finite and Deterministic DEVS Networks". *IEEE Transactions on Automation Science and Engineering* 6(3): 468–478.

Kurt, J. and L. M. Kristensen. 2009. *Coloured Petri nets: modelling and validation of concurrent systems*. 1st ed. Berlin Heidelberg: Springer-Verlag Berlin Heidelberg.

Kurt, J., S. Christensen, and L. M. Kristensen. 2006. "*State Space analysis Manual*". Denmark: Aarhus.

Kasputis, S. and H. C. Ng. 2000. "Composable simulations". In *Proceedings of the 2000 Winter Simulation Conference*, edited by J. A. Joines, R. R. Barton, K. Kang, and P. A. Fishwick. 1577–1584. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Mahmood, I. 2013. "*A Verification Framework for Component Based Modeling and Simulation - Putting the pieces together*". Stockholm: KTH Royal Institute of Technology.

Mahmood, I., R. Ayani , V. Vlassov, and F. Moradi. 2012. "Verifying Dynamic Semantic Composability of BOM-Based Composed Models Using Colored Petri Nets". In *the 26th Workshop on Principles of Advanced and Distributed Simulation*, July 15th – 19th, China, 250–257.

Mahmood, I., R. Ayani, V. Vlassov, and F. Moradi. 2011. "Fairness Verification of BOM-Based Composed Models Using Petri Nets". In *Workshop on Principles of Advanced and Distributed Simulation (PADS)*, June14th–17th, Nice, France: IEEE. 1-8.

Mahmood, I., T. Kausar, H. S. Sarjoughian, A. W. Malik, and N. Riaz. 2019. "An Integrated Modeling, Simulation and Analysis Framework for Engineering Complex Systems". *IEEE Access* 7:67497–67514.

Mayer, G. R. , and H. S. Sarjoughian. 2009. "Composable Cellular Automata". *SAGE Transactions of The Society for Modeling and Simulation International* 85(11–12):735–749.

Medjahed, B., and A. Bouguettaya. 2005. "A Multilevel Composability Model for Semantic Web Services". *IEEE Transactions on Knowledge and Data Engineering* 17(7): 954–968.

Moradi, F., R. Ayani, S. Mokarizadeh, G. H. A. Shahmirzadi, and G. Tan. 2007. "A Rule-based Approach to Syntactic and Semantic Composition of BOMs". . In *Proceedings of the 11th IEEE International Symposium on Distributed Simulation and Real-Time Applications (DS-RT'07)*, October 22nd –26th, Chania, Greece, 145–155.

Pace, D. K. 2004. "Modeling and simulation verification and validation challenges". *Johns Hopkins APL Technical Digest* 25:163–172.

Petri, C. A., and R. Wolfgang. 2008. *Petri net* 3:6477. Boston, MA: Scholarpedia.

Petty, M. D. 2008. "*Verification and Validation*". In *Principles of Modeling and Simulation: A Multidisciplinary Approach*, edited by John A., C. M. Banks and J. Sokolowski, 121–149. New Jersey: Wiley & Sons, Inc.

Platzer, A. 2008. "Differential Dynamic Logic for Hybrid Systems". *Journal of Automated Reasoning* 41:143–189.

Sargent, R. G. 2005. "Verification and validation of simulation models". In *Proceedings of the 2005 Winter Simulation Conference*, edited by M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, , 12–24. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

Sarjoughian, H. S. 2006. "Model Composability". In *Proceedings of the 2006 Winter Simulation Conference*, edited by L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, 149–158. Piscataway, New Jersey: Institute of Electrical and Electronics Engineers, Inc.

SISO. 2007. "IEEE Recommended Practice for Verification, Validation, and Accreditation of a Federation". Simulation Interoperability Standards Organization.

Tolk, A. 2010. "*Interoperability and composability*". In *Modeling and Simulation Fundamentals*, edited by Sokolowski J. A. and C.M. Banks, 403–434. New Jersey: Wiley & Sons, Inc.

Wymore, A. W. 1993. *Model-Based Systems Engineering*. USA: CRC Press, Inc.

Xia, M., K. Lo, S. Shao, and M. Sun. 2013. "Formal Modeling and Verification for MVB". *Journal of Applied Mathematics* 2013: 1–12.

Zeigler, B. P., H. Prähofer, and T. G. Kim. 2000. *Theory of Modeling and Simulation*. 2nd Edition, Academic press

Zhu, Z., Y. Lei, A. Alshareef, H. S. Sarjoughian, and Y. Zhu. 2018. "Domain Specific Meta Modeling for Deep Semantic Composability". *IEEE Access* 6:18276–18289.

## AUTHOR BIOGRAPHIES

**IMRAN MAHMOOD** is currently working as a Senior Research fellow at the University of South Florida. He has served as research fellow at Brunel University London, and as an Assistant Professor at the Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, (Pakistan). Imran earned his Masters and doctoral degrees in Computer Systems at the School of Information and Communication Technology (ICT), KTH-Royal Institute of Technology Sweden in 2007 and 2013 respectively. He is serving as the Director of the Center for Research in Modeling and Simulation (CRIMSON). His current research interests are in applied modeling, simulation, analysis and formal verification of complex systems. He can be reached at imran.mahmood@seecs.edu.pk.

**SYED HASSAN ASKARI** is currently working as a Lecturer at the Department of Software Engineering, The University of Lahore, (Pakistan). His current research interests are Model Composability and Colored Petri Nets. He can be reached at saskari.msit15seecs@seecs.edu.pk.

**HESSAM S. SARJOUGHIAN** is an Associate Professor of Computer Science and Computer Engineering in the School of Computing, Informatics, and Decision Systems Engineering (CIDSE) at Arizona State University (ASU), Tempe, AZ, and co-director of the Arizona Center for Integrative Modeling & Simulation (ACIMS). He is a core faculty in the School of Complex Adaptive Systems at Arizona State University. His research interests include model theory, poly-formalism modeling, collaborative modeling, simulation for complexity science, and M&S frameworks/tools. He can be contacted at hessam.sarjoughian@asu.edu.