An Integrated Modeling, Simulation and Analysis Framework for Engineering Complex Systems

Article *in* IEEE Access · May 2019

DOI: 10.1109/ACCESS.2019.2917652

Imran Mahmood
KTH Royal Institute of Technology
21 PUBLICATIONS **58** CITATIONS
SEE PROFILE

Hessam Sarjoughian
Arizona State University
222 PUBLICATIONS **1,849** CITATIONS
SEE PROFILE

Asad Malik
National University of Sciences and Technology
71 PUBLICATIONS **341** CITATIONS
SEE PROFILE

IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received April 21, 2019, accepted May 11, 2019. Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2019.2917652

An Integrated Modeling, Simulation and Analysis Framework for Engineering Complex Systems

IMRAN MAHMOOD[1], TAMEEN KAUSAR[1], HESSAM S. SARJOUGHIAN[2], ASAD WAQAR MALIK[1,3], AND NAVEED RIAZ[1]

[1]Center for Research in Modeling and Simulation (Crimson), School of Electrical Engineering and Computer Science (SEECS), National University of Sciences and Technology (NUST), Islamabad 44000, Pakistan

[2]Arizona Center for Integrative Modeling and Simulation (ACIMS), School of Computing, Informatics, and Decision Systems Engineering, Arizona State University, Tempe, AZ 85281, USA

[3]Department of Information Systems, Faculty of Computer Science and Information Technology, University of Malaya, Kuala Lumpur 50603, Malaysia

Corresponding author: Imran Mahmood (imran.mahmood@seecs.edu.pk)

ABSTRACT The discipline of component-based modeling and simulation offers promising gains in reducing cost, time, and the complexity of model development through the (re)use of modular components. Model-driven development suggests 1) the realization of a complex system using a conceptual model; 2) its automatic transformation into an executable form using transformation rules, and; 3) its automatic verification using a formal analysis technique for an accurate assessment of its correctness. Both approaches have numerous complementary benefits in rapid prototyping of complex systems using model reuse. In this paper, we propose a framework grounded in a combination of component-based and model-driven approaches to promote rapid prototyping of complex systems through the effective reuse of the simulation models. Our proposed process allows developers to 1) build or select existing components and compose them to formulate the conceptual models of complex systems; 2) automatically transform the conceptual models for the rapid implementation and simulation, and; 3) automatically verify them as per the requirement specifications. We propose the use of the extended finite-state machine (EFSM) as conceptual modeling formalism, anylogic simulation platform for the implementation, and probabilistic model checking technique using communicating sequential process (CSP) formalism for the verification. Finally, we present a case study of a real-time adaptive cruise control system to demonstrate the functionality of our framework. Our proposed component-based model-driven approach facilitates rapid prototyping and effective meaningful reuse of complex system models, which further accelerates the modeling, simulation, and analysis process of real-time systems and aids in complex engineering designs and implementations.

INDEX TERMS Component-based development, model driven engineering, complex systems, anylogic simulation, probabilistic model checking, adaptive cruise control system.

I. INTRODUCTION

Modeling & Simulation (M&S) provide essential means to guide the design of complex engineering systems and aid in dealing with the increasing complexity involved in their development. This growing complexity necessitates advances in the discipline of M&S. Modeling concerns with the building of a model, through the abstraction of the real system. A simulation, on the other hand, is the implementation of the model, in an executable environment, to imitate the operations of the real system, over a period of time and to generate an artificial history of the system, which is observed to draw conclusions about it. According to the definitions of the U.S. Department of Defense:

Model: "*a physical, mathematical, or otherwise logical representation of a system, entity, phenomenon, or process*".

Simulation: "*a method for implementing a model and behaviors in executable software*" [1].

Modeling real-world complex systems is a challenging task. Complex systems generally have three attributes: (i) A complex system has many parts (or units, individuals or subsystems); (ii) There are many relationships, interactions, dependencies or competitions between these parts; (iii) The parts produce combined effects (emergence) that are not easily foreseen and may often be novel (desirable) or chaotic (undesirable) [2]. Some examples of

The associate editor coordinating the review of this manuscript and approving it for publication was Giambattista Gruosso.

complex systems include, but are not limited to: Earth's atmosphere, organisms, cell biology, human brain, ecosystems, economic systems, epidemics, infrastructure such as power grid, energy markets, transportation systems, communication systems, and the World Wide Web. Main elements of complexity in these complex systems include: emergence, self-organization, collective behavior, networks, evolution & adaptation, pattern formation, nonlinear dynamics and nonintuitive system behavior [3]. A major challenge is to formulate an appropriate conceptual model that allows to represent the static structure and the dynamic behavior of a complex system. This include expressing the relevant attributes, functions and the states of the system's entities, their inputs and outputs with interconnections, and their interfaces to the exogenous systems. Formulating such constructs together into an effective, validated, and verified model can be daunting. Many real-world complex systems exhibit real-time, reactive and probabilistic behavior. In real-time systems the behavioral correctness depends on the logical results as well as the individual and the collective physical time constraints for accessing, processing, and delivering the computed information [4]. Reactive behavior means the system responds or reacts to the external events and makes progress to fulfill desired goals [5]. Probabilistic systems exhibit non-deterministic behavior for modeling random phenomena [6]. A combination of all these characteristics makes a system highly complex and difficult to model. Implementing these models for simulations, and Verification & Validation for ensuring their correctness add further intricacies in the process of complex system engineering. In this paper, we focus on the model-based development of engineered complex systems.

A. COMPONENT BASED DEVELOPMENT

The discipline of Component-based development has been identified as a key enabler in the design and development of complex systems through the (re)use of prefabricated model components [7]. The extensive use (and reuse) of modular components, offers numerous advantages including reduction in development cost, time, and system complexity and allows logical partitioning of complex systems [8]. A 'model component' is an independent building block that conforms to a standard, has well defined interfaces (inputs/outputs), functionality (process) and the dynamic behavior (states). The interfaces describe its communication with other components whereas its internal behavior is specified using a formal specification [9]. The behavior of a system is usually described using a formal specification such as: Finite state machines, Timed Automata, Petri Nets, communicating sequential processes (CSP), or similar formalism depending on the nature of the system being modeled. In this paper, we choose Extended Finite State machines [10] for the formal description of the model components. As opposed to conventional finite state machine, an EFSM captures complex behavior with more expressive transitions including: events, trigger conditions, triggered actions, and data operations on the internal variables. EFSM is also suitable for distributed and concurrent systems.

A model component is not built as a standalone element, but can be independently deployed, and it is subject to third party composition with or without modifications. The model components are composable if their inputs match the outputs. However, in order to build a meaningful composition, such that they will perform correctly according to the desired requirements, their composition is verified at a semantic level. This requires an in depth analysis of 'model composability' [11]. Composability was identified as a key objective and a daunting research challenge in early 2000 and is still being described as "biggest simulation challenge" [12]. Model composability is the capability to select and assemble model components in various combinations to satisfy specific user requirements [13].

We consider Composability as a problem of model verification, because it is the process of determining the correctness of a model with respect to its specifications [14]. Verification is concerned with the correctness of the model, i.e., a model with no errors, bugs or defects in its specification, design, and implementation [15]. Correctness of a composed model is therefore relative to its specifications, as its constituting components are required to possess precise structural and behavioral properties in order to fulfill given requirements. Requirement specification are the set of property, representing the goals and or the constraints of the model that must be fulfilled. Goals of the model can be considered as the final state(s) of the composed model or the desirable outputs produced collectively by the composed model which cannot be produced alone by the individual components. Similarly, the property constraints are the system properties that must be satisfied [16]. In this paper we present the use of Composability Verification approach for evaluating the correctness of a composed model with respect to its requirement specification using model checking to show that the selected components are composeable at dynamic level.

B. MODEL DRIVEN ENGINEERING

Model driven engineering (MDE) is a software development methodology [17] that focuses on the development of conceptual models, which mainly aim at the abstract structural and behavioral representations of a particular application domain, and automatic realization and deployment of these conceptual models into the executable models through rule based transformations [18]. A conceptual model is an abstraction of a real system and is expressed using a formal specification, based on given requirements and modeling objectives. It is later implemented and deployed in the form of a concrete computer program called executable model [19]. Our proposed framework is based on three types of MDE — Meta models:

1) CONCEPTUAL MODEL

Formal specification is used to express the structural and behavioral syntax, semantics and pragmatics of complex systems. In this paper, we use Extended Finite State machine (EFSM) formalism as a conceptual modeling framework to define model components and their composition [10]. A brief description of EFSM formalism in given in Appendix-A. EFSM is a powerful method of describing the behavior of a complex system with real-time specifications, conditional transitions, variable manipulations and data operations. EFSM is used to best capture the additional modalities of the structure and behavior of a system, which are not available in ordinary finite state-machines. It models a system at a higher-level of abstraction than a program and may contain many interleaved computations. Therefore it makes sense to decompose the system's complexity into modular parts and recombine them in a hierarchal fashion, in order to achieve reduction in logical complexity and an increase in the reuse [20]. We propose the use of EFSM formalism as our *Conceptual Model Specification*, where an EFSM-based model component encapsulates its structure and behavior using a set of states, transitions, guards, actions and state variables; and exposes its interfaces through a set of send/receive events. A set of model components communicate with each other through these interfaces and hence are composed together to form a meaningful whole system.

2) EXECUTABLE MODEL

We use Anylogic simulation platform for the executable deployment of EFSM models. A brief description about the Anylogic Simulation environment is given in Appendix-B. We propose a transformation tool to automatically transform individual EFSM components into executable specifications of Anylogic platform, while preserving the pattern of their composition. Once the EFSM-based conceptual model is completely transformed into an executable model, the modelers can simulate it using Anylogic Integrated development environment (IDE) or in standalone run time environment, and visualize the simulation results. This step allows the modeler to validate their conceptual model by comparing the simulation results with the results of the real system.

3) VERIFICATION MODEL

For the purpose of analyzing the correctness of a composed model, it is desirable to transform it into a specification that is compliant with the targeted model verification approach. In order to verify composability at this level we propose the use of Probabilistic Model Checking technique using Communicating Sequential Process (CSP) formalism [21] [22], a model description language for our *Verification Model specification,*which is executed and verified in Process Analysis Toolkit (PAT) [23], and propose an automated transformation of model components into CSP, for formal verification. A brief overview of Modeling Checking and primitives used in CSP formalism are described in Appendix-C.

C. COMPONENTBASED MODEL DRIVEN APPROACH

In this paper we propose a rapid prototyping framework for engineering complex systems. It combines the benefits of Component-based development and Model driven approach. The basic idea is to allow modelers to develop and reuse model components using a specification formalism. In this paper, we propose EFSM as a specification formalism. However, more formalisms such as DEVS, Petri Nets, SysML can be used to extend our approach. Modelers build and store these model components in a repository. When needed, they can search, discover, match and compose required components to form a conceptual model as per given requirements. The details of discovery, matching and composition (DMC) paradigm are given in [9]. When a conceptual model is formulated it is automatically transformed into an executable model and is deployed on an executable platform. We propose the use of Anylogic Simulation Platform [24], being an industrial standard, with a robust run time environment and support of event driven, real time, probabilistic and reactive modalities. During this transformation, the elements of each model component such as: states, transitions, guards, events, actions, data variables are translated into the corresponding elements of the Any logic primitives using transformation rules. The composition pattern and the coupling of the model components is also translated into the ports and connections in Anylogic. Once the transformation is complete, an Anylogic file is generated which can be either opened in the Anylogic IDE or can directly be executed using Anylogic standalone run time. A modeler can simulate it and view the simulation results. To the best of our knowledge, a run time environment for executable EFSM has not been implemented before and therefore our contribution may help the community revive the use of EFSM formalism in modeling, simulation and verification of complex systems. In order to verify the correctness of the conceptual model, we propose the use of Model checking approach using Process Analysis Toolkit [23], which uses Communicating Sequential Processes (CSP) as the verification model formalism. We choose this platform because PAT toolkit supports probabilistic verification. It also allows the use of time constraints for time-based probabilistic systems. It is to note that classical EFSM specification doesn't include the notions of time, therefore we propose an extension to include time constructs for allowing modelers to conceptualize time-based systems. When the model is transformed into CSP specification, it can be verified using PAT model checker using Probabilistic Linear Temporal Logic (PLTL) assertions. These assertions are taken as input from the modelers for verifying the model composability.

Lastly, we provide a case study of an adaptive cruise control system, as an example of a real-time nondeterministic reactive system for the proof of concept of our proposed approach. Our case study explains the process of: (i) Developing and composing EFSM based conceptual model; (ii) Transforming the conceptual model into an executable model and its simulation using Anylogic runtime environment; (iii) and verifying the composed model using model checking technique, with respect to given requirement specifications. A verified composed model conforms to its requirement specifications by successfully reaching its goal state(s) and by satisfying given constraints. Hence, we can easily say that a verified composed model developed through the systematic process of model driven engineering, asserts meaningful reuse of model components, helps reduce cost, time and complexity of model development and therefore achieves the goal of rapid prototyping of complex systems. It further allows domain experts to focus more on the domain specific conceptual modeling instead of dealing with the intricacies of the platform specific code implementations and therefore increase the productivity in Modeling & Simulation community.

The rest of the paper is organized as follows: Section 2, briefly discussed the background concepts and definitions used in this paper and provides a literature survey; Section 3 describes our proposed framework; Section 4 provides a case study of an Adaptive Cruise Control System and finally; Section 5 frames the summary, conclusion & the future work.

II. RELATED WORK

In this section, we discuss the literature review, divided in the following categories:

A. COMPONENT-BASED APPROACHES

A variety of component-based approaches such as Discrete Event System Specification (DEVS) [25] and Petri net [26] can lend themselves to specify and simulate system structure and behavior in terms of modular, hierarchical component-based models. Similarly, methods such as Linear Temporal Logic (LTL) [27] and Timed Automata (TA) [28] can specify property constraints that can be verified. Ongoing research also places strong focus on mixed continuous and discrete models (e.g., [29], [30]). These works share key common concepts and methods with those examined next with respect to the proposed Component-based Model Driven Approach.

B. MODEL DRIVEN ENGINEERING

A ''value-driven engineering'' approach is proposed in the simulation process, where the design concept along the lifecycle is aggregated to a value function. This multi-level simulation framework links Anylogic simulation platform with CAD modeling tools [31]. Another approach presents an automatic transformation from the UML activity diagrams to AnyLogic in order to generate an executable simulation model, that identifies performance issues early in an engineering design process [32]. Another approach presents the practices of three large industrial participants and proposes the use of MDE on the development of large and complex systems. MDE has been presented useful for the abstractions of complex systems at varied perspectives in order to build domain specific models, for the simulation, testing, and the analysis of performance-related decision support. Anylogic simulation software has been used in one of the case studies [33].

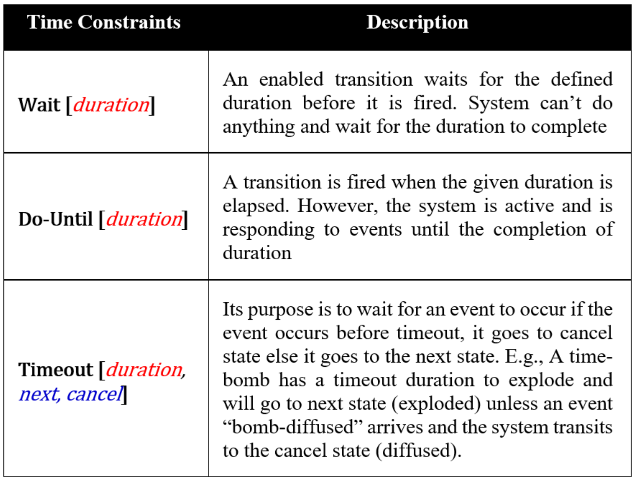
C. FORMAL SPECIFICATIONS AND VERIFICATION

Some works have been aimed at the use of formal specifications for both simulation and model-checking. Examples include transforming RTA-DEVS to TA [34]. The goal is to verify DEVS models through manually transforming them to a subset of TA which can then be executed in UPPAAL. In another work, Finite-Deterministic DEVS models are mapped into non-deterministic automata. Then, the DEVS/MS4 Me tool [34] is used for validation and the SPIN/PROMELA tool [35] is used for verification. These works, as in the proposed framework in this paper, employ different modeling methods with manual model transformations and separately developed tools. Considering Petri net, the bounded Read Arc Timed Petri net capable of checking the presence of tokens without consuming them has been developed [36]. This variant of Petri net is computationally equal to Timed Automata. The underlying tactic behind these works is to support design through loosely related simulation and model-checking methods. Although the proposed framework has the same aim as the others, it provides the EFSM and LTL models with model transformations. These modeling methods offer an alternative framework supported with the Anylogic [24] and Process Analysis Tool (PAT) [23] tools. A similar work focusses on the modeling and verification of hybrid systems, using Hybrid Communicating Sequential Processes (HCSP) [37]. Another work focuses on the translation of Simulink/State flow-based control-oriented block-diagram formalism into the hybrid-state process calculus HCSP, for full formal verification using the Isabell/HOL-based interactive verifier [38]. In contrast to the above, a unified modeling framework has been proposed where both simulation and model-checking share a common mathematical formalism [39]. In this approach, the parallel DEVS simulation models are systematically augmented so that desired properties can be specified. To support verification, the Constraint DEVS along with experiments allow defining and testing properties. A verification algorithm capable of verifying stochastic properties of the Constrained DEVS models has been developed. The verification is achieved by DEVS transducer models that evaluate the verification models that are subject to a finite set of inputs from DEVS generator models. Model-checking specification with an execution protocol is supported in the DEVS-Suite simulator [40], thus allowing unified model validation and verification in a single framework. In contrast to this work, the proposed Component-based Model Driven Approach offers a strong separation between simulation and model-checking as detailed next.

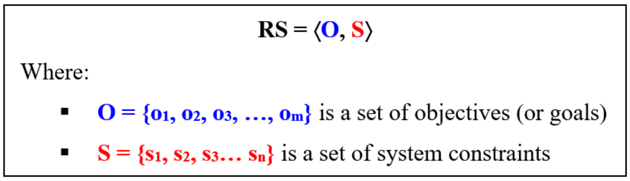
III. COMPONENT BASED MODEL DRIVEN APPROACH

This section discusses the salient features of our proposed framework presented in this paper. In this section, we describe

TABLE 1. Proposed time constraints for EFSM.



Time Constraints	Description
Wait [*duration*]	An enabled transition waits for the defined duration before it is fired. System can't do anything and wait for the duration to complete
Do-Until [*duration*]	A transition is fired when the given duration is elapsed. However, the system is active and is responding to events until the completion of duration
Timeout [*duration, next, cancel*]	Its purpose is to wait for an event to occur if the event occurs before timeout, it goes to cancel state else it goes to the next state. E.g., A time-bomb has a timeout duration to explode and will go to next state (exploded) unless an event "bomb-diffused" arrives and the system transits to the cancel state (diffused).



$$RS = \langle O, S \rangle$$

Where:

- $O = \{o_1, o_2, o_3, \ldots, o_m\}$ is a set of objectives (or goals)
- $S = \{s_1, s_2, s_3 \ldots s_n\}$ is a set of system constraints

FIGURE 1. Requirement specification template.

our proposed component-based model driven framework which consists of the phases, as shown in Figure 2.

In this paper, we focus on the model development of real-time complex systems, therefore further propose a set of time constraints as extensions in the modeling elements of EFSM formalism to allow the modelers to express time elements and constraints in EFSM models. We define three types of time constraints, which can be assigned to a transition as shown in Table 1.

A. PHASE-I: DEFINING THE SIMULAND

In the first phase, a Simuland is given as input. *Simuland is* the body of knowledge of the real system that is to be simulated [13]. We assume the modelers use any informal technique to describe the Simuland, for example natural language or UML diagrams.

B. PHASE-II: REQUIREMENT SPECIFICATION

When the Simuland is acquired, it is used to initiate the process of requirement engineering for the formulation of the requirement specifications. In this paper, we consider requirement specifications as a set of goals and constraints shown in Figure 1.

In modeling terms, an objective $\mathbf{o_i} \in \mathbf{O}$ is defined as a property that represents certain "Reachable final state", a desirable output or an emergent effect produced by the composed model or any of its components. A system constraint $\mathbf{s_i} \in \mathbf{S}$ is defined as a property that must be satisfied (or falsified), e.g., a good state; which must be reached or a bad state; which must be avoided (never be reached) during the execution. For details and examples readers are referred to [16].

C. PHASE-III: CONCEPTUAL MODEL

Conceptual Model is developed through the use and reuse of EFSM based components. We propose an XML notation to define components in EFSM format. Another XML notation is proposed to define the EFSM component composition. The schema and examples of our XML notations are available at GitHub.[1]

1) DISCOVERY, MATCHING & COMPOSITION (DMC)

A modeler defines basic components and store them in a repository. These components are searched using the information given in the simuland. When the candidate components are discovered, they are matched and filtered by the modelers. The components are compared from the simuland and requirement specifications and the most suitable selection of the candidate components are picked.

If the required components are not discovered and do not exist in the repository, then they are constructed from the scratch and are ingested in to the repository for later use. Figure 3 illustrates the DMC process. Similar approaches of DMC paradigm and the semantic discovery, matching and composition of components have been proposed [41]–[43].

D. PHASE-IV: EXECUTABLE MODEL

A conceptual model is only a formal representation of a real system. In order to be able to develop its implementation and simulate it, we need to transform the conceptual model into an executable model. In this paper, we propose to use the runtime environment of Anylogic simulation software for the implementation of the EFSM formalism. We have developed a transformation tool that takes each member EFSM component of the conceptual model as input, parse its XML and transform it into a corresponding implementation using Anylogic code snippet [44]. When all the member components of the XML EFSM are transformed, they are composed together using the Anylogic ports and connectors, according to the composition pattern given in the XML specification of the conceptual model. When the transformation process is successfully completed, the complete Anylogic file (.alp) is generated which can be opened using Anylogic IDE and can be executed. Our transformation tool uses the elements of EFSM and transform them into Anylogic elements as shown in Table 2.

E. PHASE-V: ANALYSIS TECHNIQUE

In this phase, the EFSM components are transformed into the corresponding CSP components. A preliminary transformation algorithm for UML state-charts is proposed in [45] which discusses the operational semantics of the transformation. We propose additional transformation rules to generate

[1] https://github.com/ModelDrivenFramework

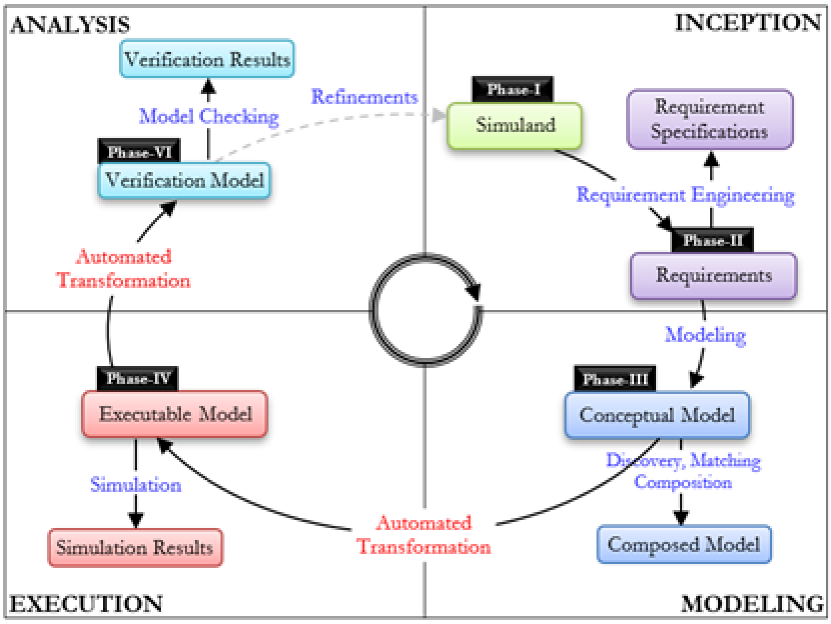


FIGURE 2. Proposed framework using component-based model driven approach.

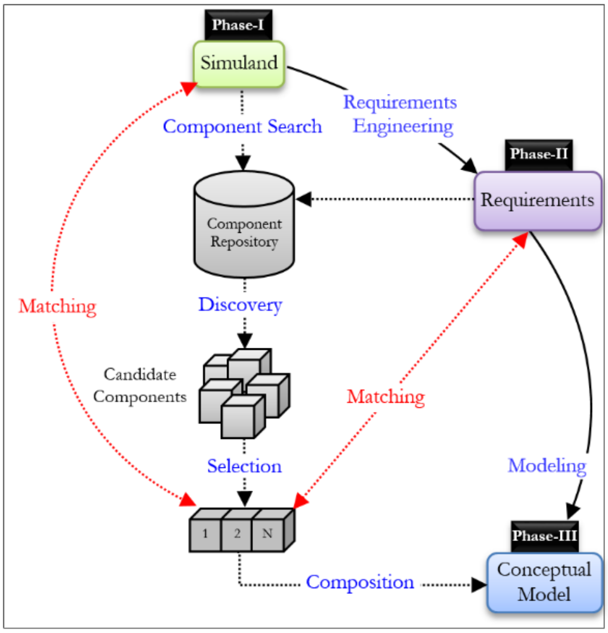


FIGURE 3. Discovery, matching & composition.

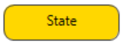
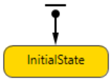
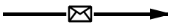
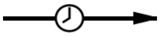
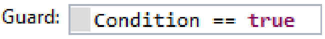
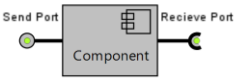
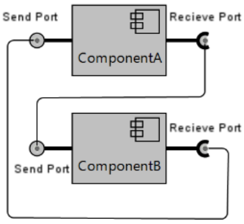
a CSP specification from EFSM as shown in Table 3. A basic example of Executable EFSM to CSP transformation is presented in Figure 4, which shows how the elements of an executable-EFSM are transformed into CSP code. We have developed a transform tool that takes each EFSM model components as input, and outputs corresponding CSP processes using PAT description language. All the CSP processes collectively represent our verification model and undergoes model checking for composability verification. The requirement specifications (objectives and constraints) are manually translated into LTL properties and passed through the model checking process for verification. If all the properties of the requirement specifications are satisfied, the model composability is evaluated as correct, and confirms the correct reuse of composed components in order to represent a real-time complex system.

1) MODEL CHECKING USING PAT MODEL CHECKER

When the Executable EFSM model is successfully transformed into CSP specification, a file is generated which can be opened in the PAT toolkit IDE. Then the properties in the requirement specifications are manually translated into *CSP# assertions,* which is a *Linear Temporal Logic (LTL), Real-Time LTL (RLTL)* and *Probabilistic LTL* (PrLTL) specification format for constructing various types of verification queries. We use PAT assertion syntax to translate the objectives and constraints of the requirement specifications, defined in Phase II. Table 4 illustrates the syntax of the CSP# assertions.

When an assertion is defined using a correct syntax, it is given as input into the PAT model checker, which in turn will present the verification results. If the assertion is satisfied the verification is successful otherwise a counter example will be highlighted, showing the exact trace where the assertion is failed. Model checking will help analyze the composability of the automatically transformed model by evaluating goal reachability and constraint satisfaction. If all the goals are reached and all the constraints are satisfied, we verify that the components used (or re-used) in the conceptual model are correctly composed, and their behaviour is consistent with respect to the requirement specification. If any counter example is observed, it will help the modelers repeat the

TABLE 2. EFSM to anylogic transformation rules.

EFSM	Anylogic Elements	Description
States (Q):		
State (q):		A state defined in Anylogic using state-chart library
Initial State (q_0)		The initial state is marked using the initial state marker
Transition (λ):		
Event (e_i) (Type = Receive)		A receive event is implemented as a message trigger using Anylogic Messaging API. It waits for a particular message expected from other components.
Event (e_i) (Type = Send)		A send event is implemented in the form of a 'time-out' trigger, with a time-out delay = 0. When it is triggered it sends a message to a corresponding recipient. A message can be of type 'String' or a complex object containing message parameters.
Guard [g]		A guard is implemented on any transition using state-chart built-in element 'Guard'. Which is a Boolean condition written in Java.
Action (a)		We implement actions using 'functions' written in Java, defined locally within the body of the EFSM component. The actions are usually use to ready input variables and write on to the output variables. The logic of the function is parsed from the source EFSM component.
Variable (V)		A set of input and output variables using bool, int, double, string or any Java compliant data type is defined within the body of EFSM component. Input variables are read when an action is triggered and the output of the action is stored in the output variables
Component(M) / Component Model:		
M_i		A component in Any logic is defined as an object. It sends messages to other components using *send port* and receives messages at the *receive port*. There is an internal queue in each component that stores the received messages in FIFO order.
$M_a \oplus M_b$	 Send Port Recieve Port ComponentA Recieve Port Send Port ComponentB	A composition in Any logic is implemented by connecting send and receive ports such that all the 'required interfaces' (receive ports) are connected to the 'provided interfaces' (send ports) of the participating components. This information is stored in the XML file of the EFSM conceptual model and is used to automatically connect respective ports.

entire process by selecting a more suitable component and therefore ensure correctness and meaningful reuse.

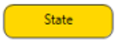
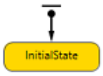
F. PERFORMANCE EVALUATION

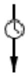
A thorough assessment and performance evaluation of our proposed framework for its ability to support industrial scale engineering prototypes of complex systems necessities an in-depth study which is beyond the scope of this paper. It not only demands a quantitative measure of the process involved, but also a qualitative metrics in terms of Adaptability and Composability degree. Adaptability is the degree to which framework can be extended to support multiple modeling, execution and verification formalisms. The degree to which the framework supports syntactic, semantic and pragmatic reuse of models in various context is called composability degree. For the purpose of the quantitative measure of the process, our framework is inherently dependent on the performance of Anylogic and PAT tool kit. The execution performance of Anylogic is further dependent on the platform being used and scales up with the migration from common desktops to high-performance computing platforms. Model Checking technique is also scalable since it relies on the usage of PAT tool which can handle about 10^7 states in a reasonable amount of time [46]. This should be sufficient for the verification of most industrial scale system models.

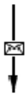
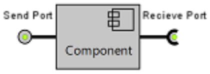
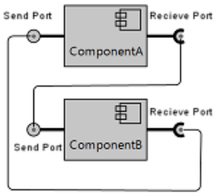
IV. CASE STUDY: ADAPTIVE CRUISE CONTROL SYSTEM MODEL

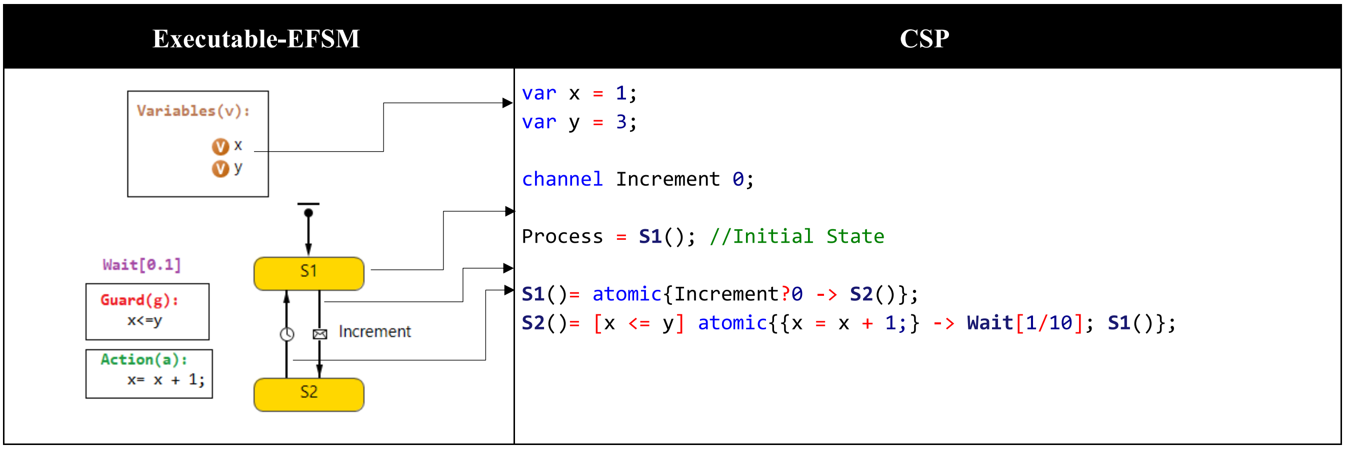
In this section we present a case study of Adaptive Cruise Control System in order to explain our proposed approach. Adaptive Cruise Control (ACC) System is a

TABLE 3. EFSM to CSP transformation rules.

Executable-EFSM	CSP	Description
States (Q):		
	State-Name();	Each state is defined as a process.
	InitialState-Name();	Initial process are initial states of the system.
Transitions (λ):		

 (Type = Send)	channel event-name 0;
	State() = [guard] Wait[d]; event ! parameters {action} ? Next-State();
		For each transition, an event channel is created in PAT. '0' represents the buffer size of the communication channel, which additionally means that it is a synchronous channel.
		A clock is defined to implement timed functions such as Wait, do-until and timeout

 (Type = Receive)	State() = [guard] Wait[d]; event ? parameters {action} ? Next-State();	The receiving transition is similarly defined except the event has a symbol '?'
Component(M) / Component Model:		
	ComponentName(i) = InitialState(i);	An initial state is defined and assigned to the component. If a component has multiple instances it is passed a parameter 'i' which represents the instance number.
 Send Port Recieve Port ComponentA Recieve Port Send Port ComponentB	Composed Model = ComponentA ||| ComponentB;	The system is defined as a composition of PAT process components with an interleaving operator '|||' between each other. In case of broadcast events (i.e., one event is sent to one to many, many to one or all components) synchronization a parallel operator '||' is used.



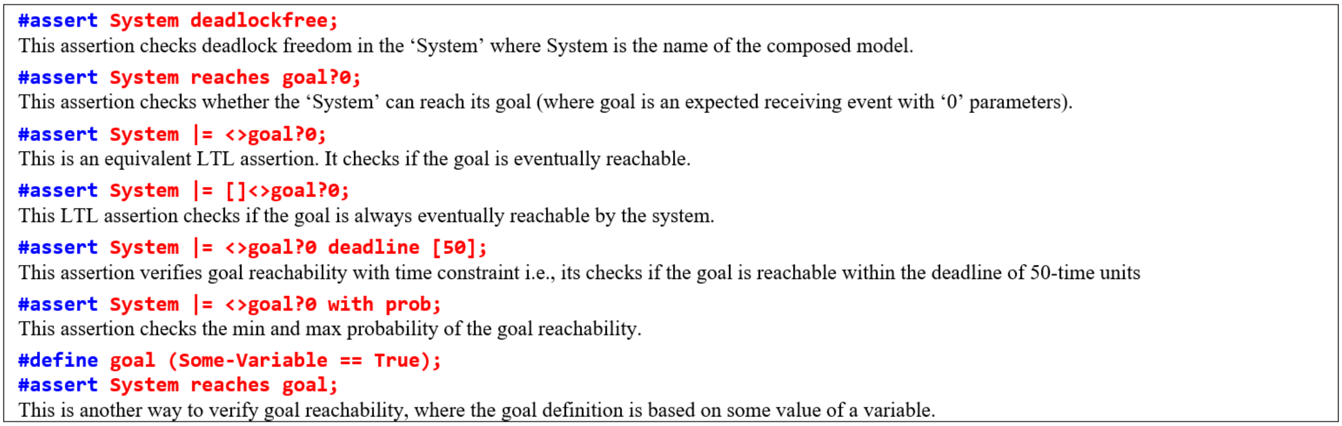
```
var x = 1;
var y = 3;

channel Increment 0;

Process = S1(); //Initial State

S1()= atomic{Increment?0 -> S2()};
S2()= [x <= y] atomic{{x = x + 1;} -> Wait[1/10]; S1()};
```

FIGURE 4. Executable EFSM to CSP transformation example.

dynamic cruise control system for automobiles, which adjusts vehicle's speed according to the traffic situation. It consists of a 'Sensor' that maintains a safe distance by detecting the speed of the leading vehicle, distance (*also called clearance*) and time gap, i.e., the time interval between the leader and ACC vehicle and is calculated as:

$$T_{gap} = \frac{Dist_{ACC}}{Speed_{ACC}}$$

TABLE 4. PAT LTL assertions.



```
#assert System deadlockfree;
```
This assertion checks deadlock freedom in the 'System' where System is the name of the composed model.

```
#assert System reaches goal?0;
```
This assertion checks whether the 'System' can reach its goal (where goal is an expected receiving event with '0' parameters).

```
#assert System |= <>goal?0;
```
This is an equivalent LTL assertion. It checks if the goal is eventually reachable.

```
#assert System |= []<>goal?0;
```
This LTL assertion checks if the goal is always eventually reachable by the system.

```
#assert System |= <>goal?0 deadline [50];
```
This assertion verifies goal reachability with time constraint i.e., its checks if the goal is reachable within the deadline of 50-time units

```
#assert System |= <>goal?0 with prob;
```
This assertion checks the min and max probability of the goal reachability.

```
#define goal (Some-Variable == True);
#assert System reaches goal;
```
This is another way to verify goal reachability, where the goal definition is based on some value of a variable.

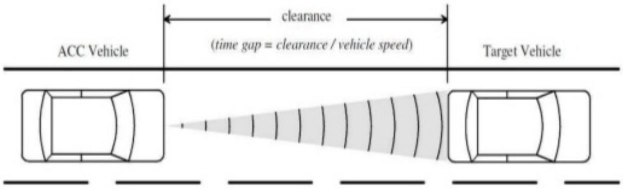
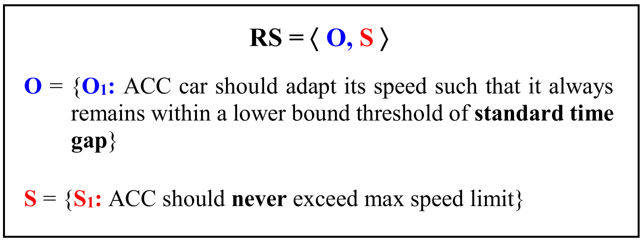


FIGURE 5. Adaptive cruise control system.



RS = ⟨ O, S ⟩

O = {O_1: ACC car should adapt its speed such that it always remains within a lower bound threshold of **standard time gap**}

S = {S_1: ACC should **never** exceed max speed limit}

FIGURE 6. ACC requirement specification.

If the sensor detects that the leading vehicle is slowing down or speeding up, it sends message to controller to communicate with actuator in order to adjust speed to maintain a standard lower bound time gap of 2 seconds to avoid collision [45].

Our task is to build the conceptual model of the adaptive cruise control real-time system using pre-developed components, and use our proposed framework to implement, simulate and model-check it. We will show how all the components work together to accomplish common goals and avoid safety constraints. In this section we present all the phases of our framework.

A. DEFINING THE SIMULAND

We define the simuland of an adaptive cruise control real-time system as follows:

a) **Sensor:** It detects the distance between two cars and communicates it to the controller in order to maintain the safe distance/time gap.
b) **Controller:** It receives periodic data from the Sensor and controls the speed of the car accordingly
c) **Actuator:** It accelerates or decelerates the speed of car as per instructions of controller.
d) **Environment:** It is the external environment of the system and is used to generate input data

B. REQUIREMENT SPECIFICATION

Based on the scenario, we define requirements of the ACC system as shown in Figure 6.

C. ACC CONCEPTUAL MODELS

Starting in phase III, we search and discover suitable EFSM components from the repository using the simuland description and requirement specifications. We matched and selected three components namely: *Sensor*, *Controller*, *Actuator*, and *Environment* from the list of candidate components.

1) SENSOR

Sensors observes the environment and sense the distance and speed of the front car. Sensor consists of two states: Receiving and Processing. Receiving is an initial state which means sensor is active and waiting to receive data from the environment. When an event '*Update*' is received, the message parameters: 'current distance' and 'speed' are obtained to calculate time gap, and the sensor jumps to the Processing State. At the processing state, if time gap is greater than a standard time gap, an '*accelerate*' event is sent to the controller. Otherwise, a '*decelerate*' event is sent. EFSM specification of the sensor is shown in Table 5.

2) CONTROLLER

The main job of the controller is to control the speed by updating the actuator, according to the data received from the sensor and ensure the satisfaction of the maximum speed limit constraint. Controller consists of three states '*active*', '*accelerating*', '*decelerating*'. '*active*' is an initial state. An event named '*accelerate*', sent by the sensor moves component from active to '*accelerating*' if the current speed is less than

TABLE 5. EFSM specification of sensor component.

$M_1 = \langle Q, \Sigma_1, \Sigma_2, I, V, \Lambda \rangle$

Q = {Receiving, Processing} **q_0:** Receiving

Σ = {update, accelerate, decelerate}

V = **V_{in}**{distance, standardTimeGap, speed}: double **V_{out}** {timeGap}:double

Λ: Transition Specifications

λ_1= **Receiving** $\xrightarrow[\{\text{distance, speed}\} \mid \{\text{timeGap}\}]{\textbf{update}\ [\ ]\ /\ \textbf{action}\ (\text{timeGap} = \text{distance/speed})}$ **Processing**

λ_2= **Processing** $\xrightarrow[\{\text{ timeGap, standardTimeGap }\} \mid \{\ \}]{\textbf{Wait}[\textbf{0.1}]\ \textbf{decelerate}\ [\text{timegap} <= \text{standardTimeGap}]\ /\ \textbf{action}\ ()}$ **Receiving**

λ_3= **Processing** $\xrightarrow[\{\text{ timeGap, standardTimeGap }\} \mid \{\ \}]{\textbf{Wait}[\textbf{0.1}]\ \textbf{accelerate}\ [\text{timegap} > \text{standardTimeGap}]\ /\ \textbf{action}\ ()}$ **Receiving**

TABLE 6. EFSM specification of controller component.

$M_2 = \langle Q, \Sigma_1, \Sigma_2, I, V, \Lambda \rangle$

Q = {active, accelerating, decelerating} **q_0:** active

Σ = {accelerate, decelerate, pedal, brake}

V = **V_{in}** {speed, maxSpeed}: double **V_{out}** {speed}:double

Λ: Transition Specifications

λ_1= **active** $\xrightarrow[\{\text{ speed, maxSpeed}\} \mid \{\ \}]{\textbf{accelerate}\ [\ \text{speed} < \text{maxSpeed}\ ]\ /\ \textbf{action}\ ()}$ **accelerating**

λ_2= **accelerating** $\xrightarrow[\{\ \} \mid \{\ \}]{\textbf{pedal}\ [\ ]\ /\ \textbf{action}\ ()}$ **active**

λ_3= **active** $\xrightarrow[\{\ \} \mid \{\ \}]{\textbf{decelerate}\ [\ ]\ /\ \textbf{action}\ ()}$ **decelerating**

λ_4= **decelerating** $\xrightarrow[\{\ \} \mid \{\ \}]{\textbf{brake}\ [\ ]\ /\ \textbf{action}\ ()}$ **active**

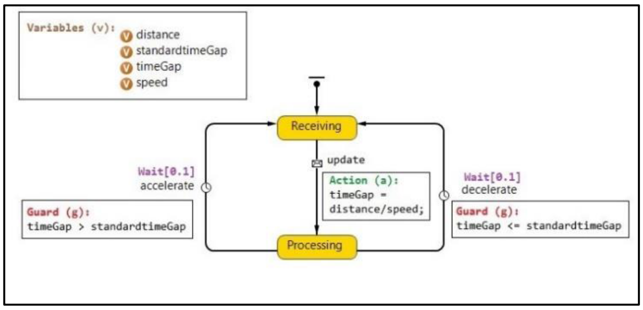


FIGURE 7. Sensor EFSM executable in anylogic.

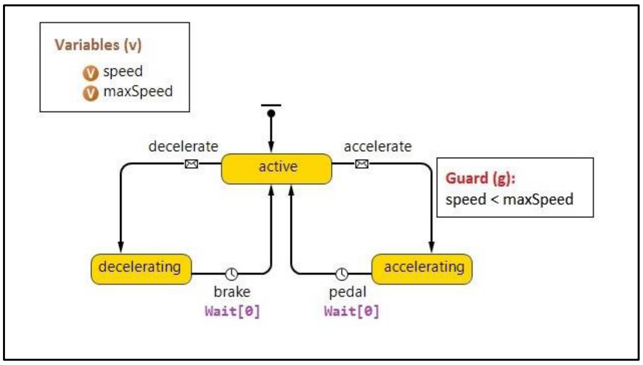


FIGURE 8. Controller EFSM executable in anylogic.

the maximum speed limit. While a '*decelerate*' event moves it from active to '*decelerating*' state. EFSM specification of the controller is shown in Table 6.

3) ACTUATOR

Actuator controls the actual speed of the car. It has three states '*fixSpeed*', '*Brake*' and '*Pedal*'. '*fixSpeed*' is an initial state of the component. On receiving '*accelerate'* event from the controller, it jumps to '*pedal'* and increment the speed by 1 unit and returns back to the '*fixSpeed*'. Similarly, on receiving '*decelerate*' it jumps to '*brake*' state and decrements the speed by 1 unit and returns. EFSM specification of the actuator is shown in Table 7.

TABLE 7. EFSM specification of actuator component.

$$M_3 = \langle Q, \Sigma_1, \Sigma_2, I, V, \Lambda \rangle$$

Q = {fixSpeed, Brake, Pedal} **q0:** fixSpeed

Σ = {brake, pedal}

$V = V_{in}$ {speed}:double V_{out} {speed}:double

Λ: Transition Specifications

$$\lambda_1 = \textbf{fixSpeed} \xrightarrow[\{\ \}\ |\ \{\ \}]{\textbf{pedal}\ [\ ]\ /\ \textbf{action}\ (\ )} \textbf{Pedal}$$

$$\lambda_2 = \textbf{Pedal} \xrightarrow[\{\text{speed}\}\ |\ \{\ \text{speed}\ \}]{\varepsilon\ [\ ]\ /\ \textbf{action}\ (\ \text{speed=speed+1}\ )} \textbf{fixSpeed}$$

$$\lambda_3 = \textbf{fixSpeed} \xrightarrow[\{\ \}\ |\ \{\ \}]{\textbf{brake}\ [\ ]\ /\ \textbf{action}\ (\ )} \textbf{Brake}$$

$$\lambda_4 = \textbf{Brake} \xrightarrow[\{\ \text{speed}\ \}\ |\ \{\ \text{speed}\ \}]{\varepsilon\ [\ ]\ /\ \textbf{action}\ (\ \text{speed=speed-1}\ )} \textbf{fixSpeed}$$

TABLE 8. EFSM specification of environment component.

$$M_4 = \langle Q, \Sigma_1, \Sigma_2, I, V, \Lambda \rangle$$

Q = {Idle} **q0:** Idle

Σ = {update}

$V = V_{out}$ {speed, distance}:double

Λ: Transition Specifications

$$\lambda_1 = \textbf{Idle} \xrightarrow[\{\ \}\ |\ \{\text{speed, distance}\}]{\textbf{Wait}[0.1]\ \textbf{update}\ [\ ]\ /\ \textbf{action}\ (\text{speed = triangulardist}(15,38,26)^2)} \textbf{Idle}$$

TABLE 9. Composed conceptual model.

$$\textbf{CM} = M_1 \oplus M_2 \oplus M_3 \oplus M_4$$

$$(M_4\,!\lambda_1 \rightarrow M_1\,?\lambda_1) + (M_1\,!\lambda_2 \rightarrow M_2\,?\lambda_1) + (M_1\,!\lambda_3 \rightarrow M_2\,?\lambda_3) + (M_2\,!\lambda_2 \rightarrow M_3\,?\lambda_1) + (M_2\,!\lambda_4 \rightarrow M_3\,?\lambda_3)$$

4) ENVIRONMENT

It is the external environment that provides random inputs: speed and distance of the front car to the sensor by sending 'update' event with parameters: speed & distance. EFSM specification of the environment is shown in Table 8.

5) COMPOSED MODEL

All the EFSM models are composed to form the conceptual model. Each component is connected to another component using '⊕' operator and the send and receive transitions are mapped using "!" send or "?" receive operators, as shown in IV-D. The detailed descriptions and the XML code of the individual EFSM components and their composed specification are available at Github.

D. ACC EXECUTABLE MODEL

In the next phase, all the EFSM components are transformed into corresponding executable models, using our transformation tool, according to the rules specified in Table 2. Finally, they are (automatically) composed together to form the executable ACC composed model. The Anylogic implementation of each component is show in Figure 7 to Figure 11:

E. ACC MODEL EXECUTION

Two scenarios of adaptive cruise control model in Anylogic were simulated: (i) Success Scenario, where the goal is achieved and the safety constraint is satisfied, showing that the selected components in the composition are consistent with respect to the requirement specification. (ii) Counter example, where the goal is not achieved, meaning one or more components are not consistent in order to achieve mutual

[2] We assume that the front car changes its speed using a triangular distribution with min=15, max = 38 and most likely =26

TABLE 10. Comparative studies.

No	Source	Technique	Min Time Gaps	Description
1	Lin et al [47]	Physical Simulator	0.64 - 2.40	A study was conducted to investigate the effects of time-gap on a fix-based bus driving simulator and on drivers' performance while reclaiming control from ACC in a car-following scenario of emergency brake by the lead vehicle.
2	Ntousakis et al [48]	Aimsun Traffic Simulator	0.8 – 2.0	This work describes the modelling of ACC-equipped vehicles in the commercial traffic simulator Aimsun. It was observed that ACC-vehicles can improve the stability of traffic, since they mitigate the intensity and number of stop-and-go waves.
3	Xiao et al [49]	Matlab Simulation	1.1	This study proposes a collision-free car-following simulation model for ACC/CACC vehicles. The testing scenarios include stop and go, approaching and cut-out maneuvers. The simulation results show that the proposed model is collision-free in the full-speed-range with the leader car.

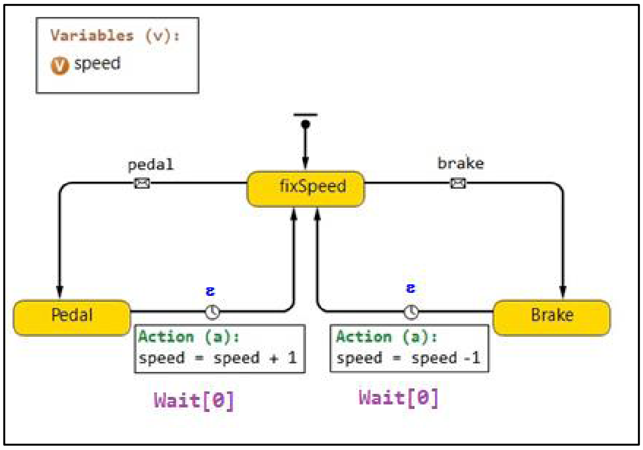


FIGURE 9. Actuator EFSM executable in anylogic.

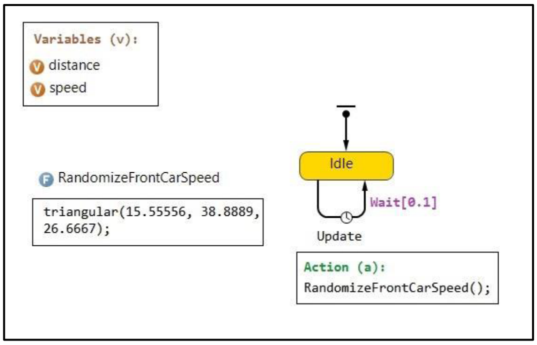


FIGURE 10. Environment EFSM executable in anylogic.

goals. In both scenarios, we assume the internal behaviour of the environment component has a leading car moving on an arbitrary random speed using Triangular Distribution [min = 15, max = 38, mode = 26] and is changing its speed at different intervals. Due to this varying speed the follower car has to adjust itself by calculating time gap and sending accelerate or brake messages to the controller.

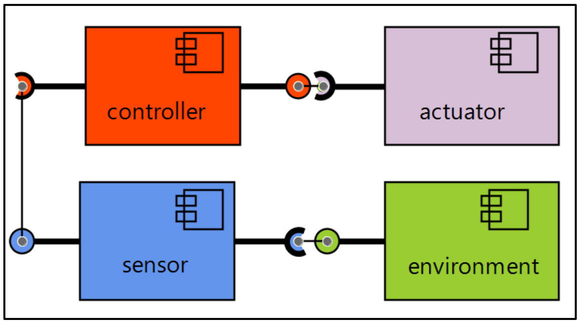


FIGURE 11. ACC composed model - executable in anylogic.

1) SIMULATION SCENARIO I

In this scenario, the selected sensor was configured at a refresh rate of 100 milli-seconds and the controller is guarded with conditions to remain within the speed limit, therefore resulting into a success scenario. Figure 12 shows the simulation results of the scenario (i).

The Figure 12 shows that the lower bound time-gap is: $T_{\mathbf{gap}} \geq 1.99$ sec and doesn't go lower than that. Since we modeled a stochastic system, each time it is simulated, it will produce different results. However, we will show in the Analysis phase that using probabilistic model checking, we can gain a confidence on the goal reachability if it is within an acceptable threshold. According to a study conducted in [47] effects of time-gap settings of ACC car were inspected. Thirty professional bus drivers drove on the simulator with the scenario of highway traffic flow under 12 random time-gap settings: from 0.64 to 2.40 seconds. The general concept is to maintain the safe time gap or safe distance from the leading car in order to avoid the collision. It was revealed that the safer time gaps for different situations were longer than **1.60 sec**. This confirms the goal reachability i.e.; *ACC car should adapt its speed such that it always remains within a threshold of standard time gap.* Similarly, figure 12 c shows that the ACC car always remain within the maximum speed limit

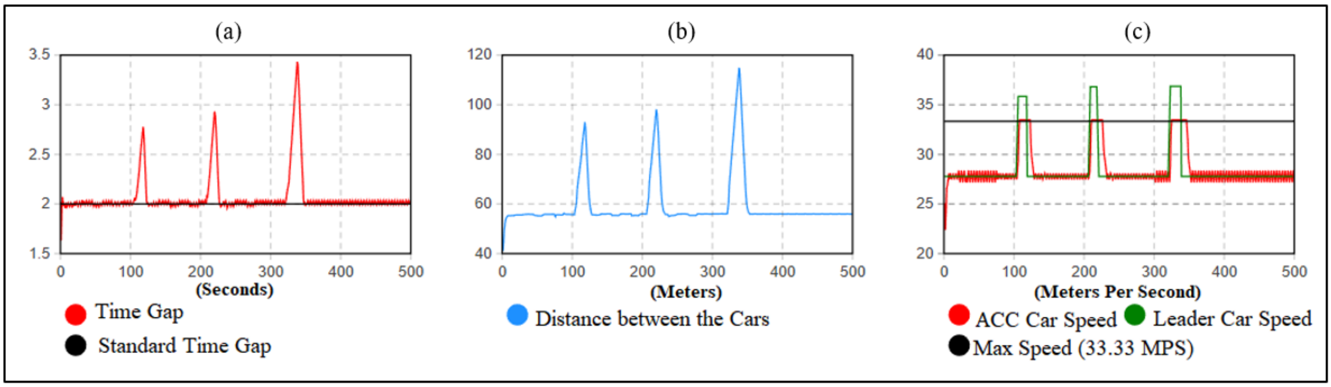


FIGURE 12. Success scenario. (a) Time gap. (b) Leader and follower distance. (c) Leader and follower speed.

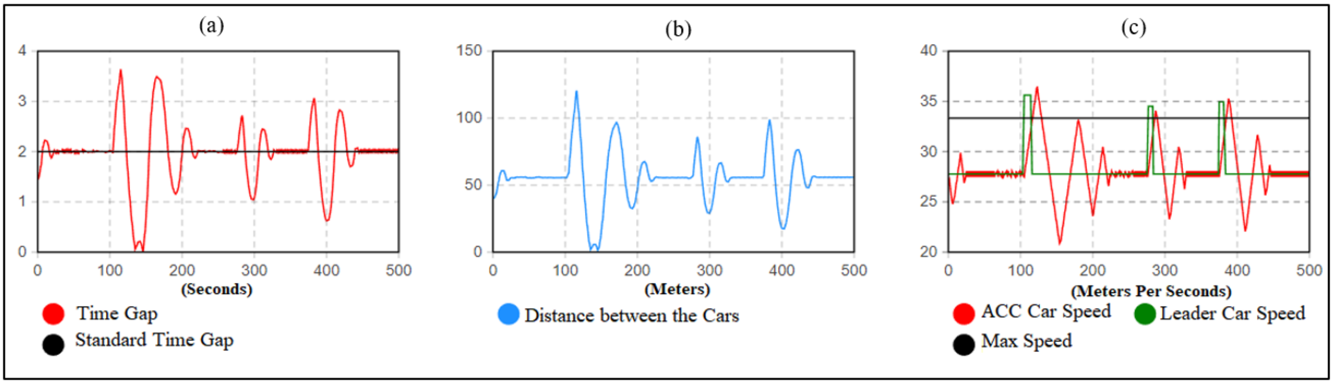
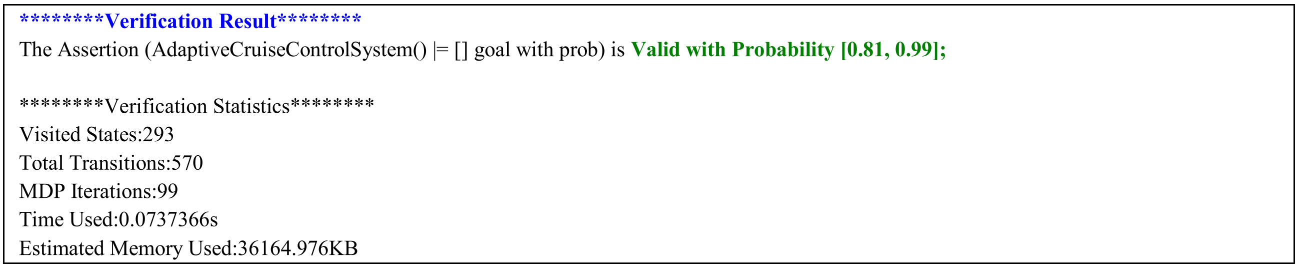


FIGURE 13. Counter example. (a) Time gap. (b) Leader and ACC car distance. (c) Leader and ACC car speed.



```
********Verification Result********
The Assertion (AdaptiveCruiseControlSystem() |= [] goal with prob) is Valid with Probability [0.81, 0.99];

********Verification Statistics********
Visited States:293
Total Transitions:570
MDP Iterations:99
Time Used:0.0737366s
Estimated Memory Used:36164.976KB
```

FIGURE 14. Goal reachability.

(i.e.., 33.3 m/s = 120 KM/Hr.) even though the speed of the leader car exceeds the limit. This shows that the satisfaction of the constraint in the requirement specification.

2) SIMULATION SCENARIO II

In this scenario, the refresh rate of the discovered sensor was 500 milli-seconds and the controller chosen was not guarded with max speed limit, thus resulted into a counter example. Figure 13 shows the simulation results of the scenario (ii). It can be noticed that the ACC car is violating the time-gap limit multiple times during the simulation, as it goes as low as 0 i.e., $T_{gap} \geq 0$, therefore it fails to reach the goal. It is also evident from Figure 13 (c) that the ACC car violates maximum speed limit (i.e.., 33.3 m/s) and therefore violates safety constraint.

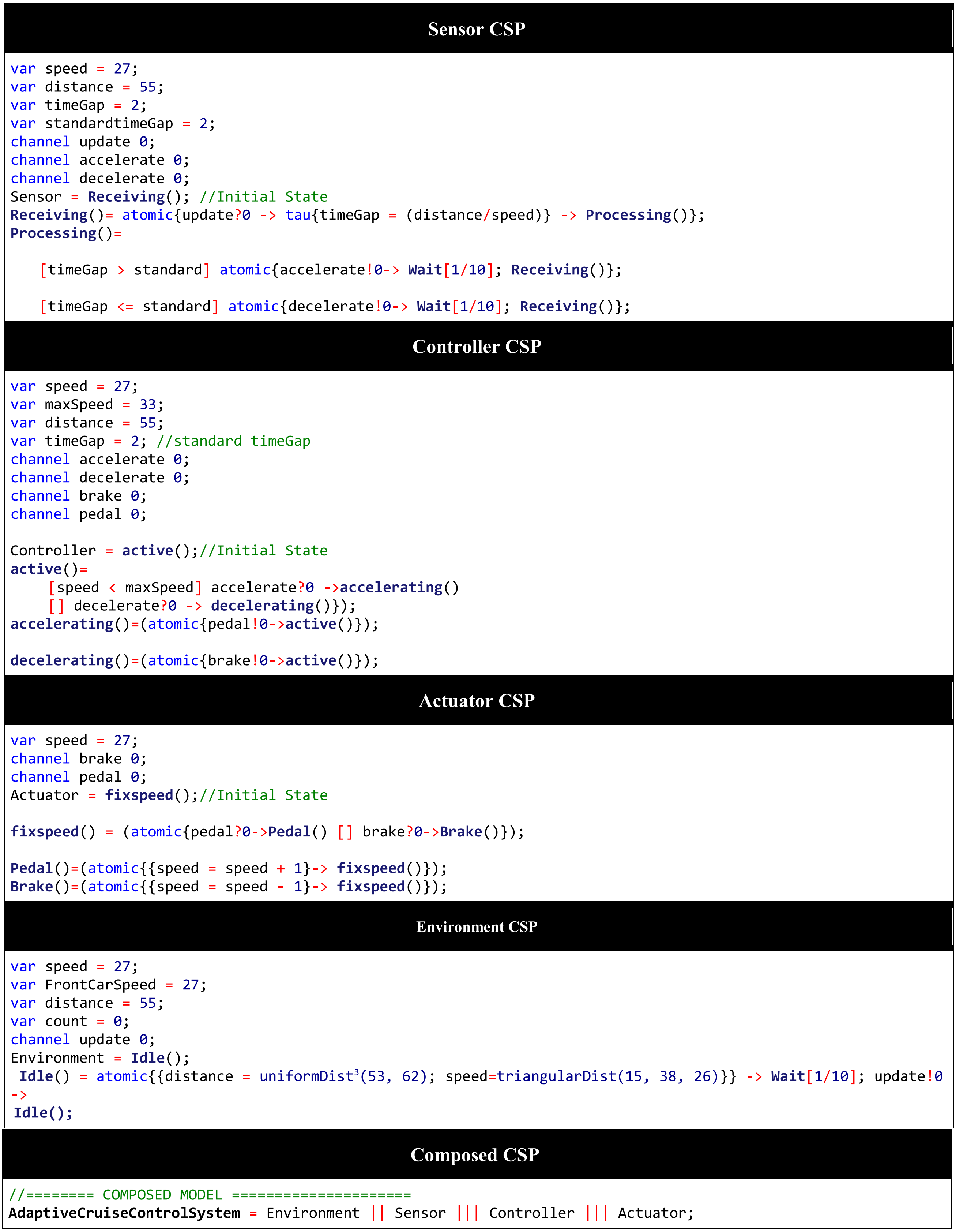
F. RESULTS VALIDATION

We compared our simulation results with the existing studies shown in Table 10. It can be seen that the time gaps computed during the simulations of these studies are comparable with the time gap presented in Figure 12.

G. ANALYSIS TECHNIQUE

In this phase we use Model Checking approach for the verification of the ACC composed model. This section presents the steps of transformation of Executable ACC Composed model

TABLE 11. Transformation of EFSM to CSP.



Sensor CSP

```
var speed = 27;
var distance = 55;
var timeGap = 2;
var standardtimeGap = 2;
channel update 0;
channel accelerate 0;
channel decelerate 0;
Sensor = Receiving(); //Initial State
Receiving()= atomic{update?0 -> tau{timeGap = (distance/speed)} -> Processing()};
Processing()=

   [timeGap > standard] atomic{accelerate!0-> Wait[1/10]; Receiving()};

   [timeGap <= standard] atomic{decelerate!0-> Wait[1/10]; Receiving()};
```

Controller CSP

```
var speed = 27;
var maxSpeed = 33;
var distance = 55;
var timeGap = 2; //standard timeGap
channel accelerate 0;
channel decelerate 0;
channel brake 0;
channel pedal 0;

Controller = active();//Initial State
active()=
    [speed < maxSpeed] accelerate?0 ->accelerating()
    [] decelerate?0 -> decelerating()});
accelerating()=(atomic{pedal!0->active()});

decelerating()=(atomic{brake!0->active()});
```

Actuator CSP

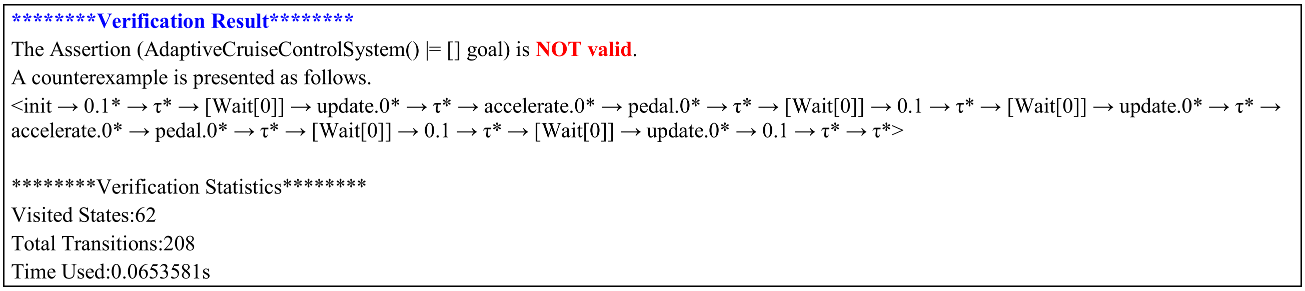
```
var speed = 27;
channel brake 0;
channel pedal 0;
Actuator = fixspeed();//Initial State

fixspeed() = (atomic{pedal?0->Pedal() [] brake?0->Brake()});

Pedal()=(atomic{{speed = speed + 1}-> fixspeed()});
Brake()=(atomic{{speed = speed - 1}-> fixspeed()});
```

Environment CSP

```
var speed = 27;
var FrontCarSpeed = 27;
var distance = 55;
var count = 0;
channel update 0;
Environment = Idle();
 Idle() = atomic{{distance = uniformDist[3](53, 62); speed=triangularDist(15, 38, 26)}} -> Wait[1/10]; update!0
->
 Idle();
```

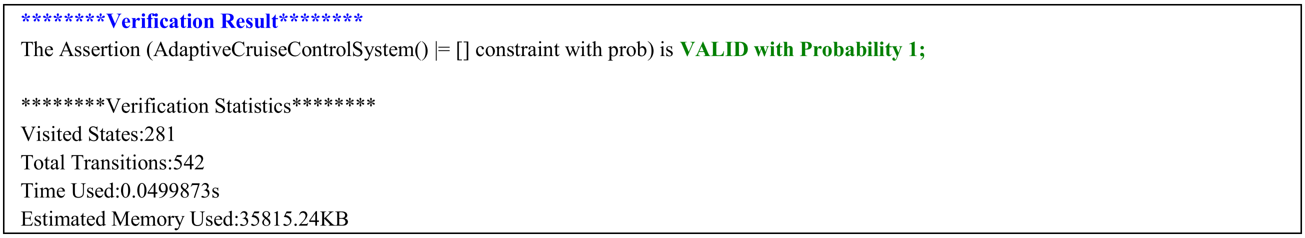
Composed CSP

```
//======== COMPOSED MODEL =====================
AdaptiveCruiseControlSystem = Environment || Sensor ||| Controller ||| Actuator;
```



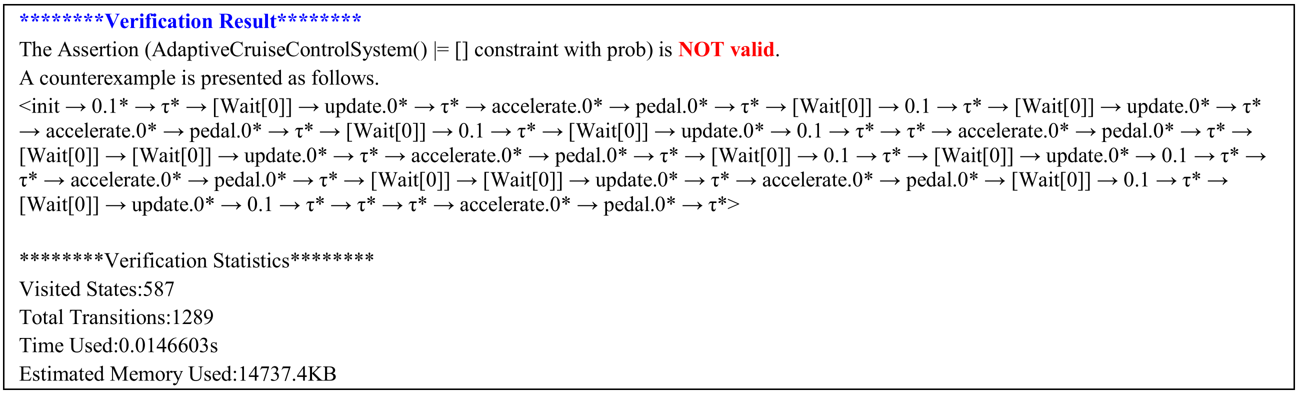
```
********Verification Result********
The Assertion (AdaptiveCruiseControlSystem() |= [] goal) is NOT valid.
A counterexample is presented as follows.
<init → 0.1* → τ* → [Wait[0]] → update.0* → τ* → accelerate.0* → pedal.0* → τ* → [Wait[0]] → 0.1 → τ* → [Wait[0]] → update.0* → τ* →
accelerate.0* → pedal.0* → τ* → [Wait[0]] → 0.1 → τ* → [Wait[0]] → update.0* → 0.1 → τ* → τ*>

********Verification Statistics********
Visited States:62
Total Transitions:208
Time Used:0.0653581s
```

FIGURE 15. Goal reachability – counter example.



```
********Verification Result********
The Assertion (AdaptiveCruiseControlSystem() |= [] constraint with prob) is VALID with Probability 1;

********Verification Statistics********
Visited States:281
Total Transitions:542
Time Used:0.0499873s
Estimated Memory Used:35815.24KB
```

FIGURE 16. Safety constraint.



```
********Verification Result********
The Assertion (AdaptiveCruiseControlSystem() |= [] constraint with prob) is NOT valid.
A counterexample is presented as follows.
<init → 0.1* → τ* → [Wait[0]] → update.0* → τ* → accelerate.0* → pedal.0* → τ* → [Wait[0]] → 0.1 → τ* → [Wait[0]] → update.0* → τ*
→ accelerate.0* → pedal.0* → τ* → [Wait[0]] → 0.1 → τ* → [Wait[0]] → update.0* → 0.1 → τ* → τ* → accelerate.0* → pedal.0* → τ* →
[Wait[0]] → [Wait[0]] → update.0* → τ* → accelerate.0* → pedal.0* → τ* → [Wait[0]] → 0.1 → τ* → [Wait[0]] → update.0* → 0.1 → τ* →
τ* → accelerate.0* → pedal.0* → τ* → [Wait[0]] → [Wait[0]] → update.0* → τ* → accelerate.0* → pedal.0* → [Wait[0]] → 0.1 → τ* →
[Wait[0]] → update.0* → 0.1 → τ* → τ* → τ* → accelerate.0* → pedal.0* → τ*>

********Verification Statistics********
Visited States:587
Total Transitions:1289
Time Used:0.0146603s
Estimated Memory Used:14737.4KB
```

FIGURE 17. Safety constraint – counter example.

TABLE 12. Assertion I.

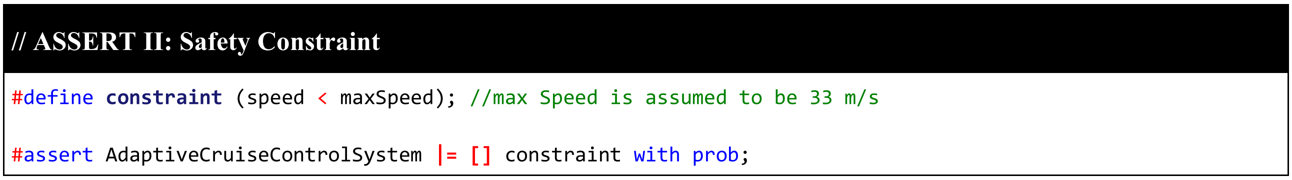
```
// ASSERT I: Goal state Reachability

#define goal (timeGap >= 2); //Standard Time gap is assumed to be 2

//The probability that the system always reaches goal
#assert AdaptiveCruiseControlSystem |= [] goal with prob;
```

into the CSP-based verification model and the formal verification using PAT analysis toolkit. The executable components are transformed using the proposed transformation rules.

The composed model is executed and verified using PAT model checker. At first, we translated the requirement specifications into PAT assertions as shown below: 'Assertion-I' uses a PLTL construct to verify that the ACC car maintains a standard time-gap. If assertion1 is satisfied, it shows that time gap between cars is standard and maintained to avoid collision. The result of PAT model checker in Figure 14 shows that the goal state is reachable as the assertion is satisfied with a confidence factor of 81% – 99%. This is not 100% because the model is

[3] The functions: ***uniformDist*** and ***traingularDist*** are not automatically transformed as there is no built-in support for these functions in PAT.

TABLE 13. Assertion II.



// ASSERT II: Safety Constraint

```
#define constraint (speed < maxSpeed); //max Speed is assumed to be 33 m/s

#assert AdaptiveCruiseControlSystem |= [] constraint with prob;
```

stochastic and on occasions the time-gap of the ACC car has dropped slightly below 2 as can be seen in Figure 12. On the other hand, the goal reachability in the counter example is failed as shown in Figure 15. 'Assertion-II' uses a PLTL construct to verify that speed of the ACC car does not exceed the maximum speed limit. The verification results of 'Assertion-II' in Figure 16 suggest the ACC will never violate the safety constraint. Whereas the results of 'Assertion-II' in the counter example are not satisfied as shown in Figure 17.

V. SUMMARY AND CONCLUSION

In this paper, we propose a framework using Component-based Model Driven Approach to promote rapid development and effective reuse of complex simulation models. Our proposed process allows developers to build, select and compose components to formulate conceptual models of complex systems; (re)use them for automatic deployment of executable simulations; and automatically verify them as per the requirement specifications. We propose the use of Extended Finite State-machine (EFSM) as conceptual modeling formalism, Anylogic Simulation platform for the executable deployment, and Model Checking technique with Communicating Sequential Processes (CSP) formalism for the verification. Lastly, we present a case study of a real-time adaptive cruise control system to demonstrate EFSM-based components for developing conceptual models, transforming conceptual models into executable Anylogic simulations, and (ii) transforming executable models into CSP for the model checking. Two scenarios of the ACC model were presented. In the success scenario the composed model was evaluated to be correct as it reached its goal, i.e., the adaptive behavior was within a given time-gap limit and satisfied its constraint, i.e., the ACC car didn't exceed its speed limit. In the counter example, both the goal and the constraint were not satisfied due to a faulty composition.

Our proposed component-based model driven approach facilitates rapid prototyping and effective meaningful reuse of complex system models. There are several benefits of our proposed framework as it uses the Discovery, Matching and Composition paradigm to construct EFSM-based conceptual models through component reuse. It provides tools for rapid implementation through automated transformation into an executable form, and allows the modeler to use Anylogic to build and simulate dynamic models followed by validating the simulation results for the given simuland. It further provides composability verification through the integration of Model Checking with Anylogic as an automated tool for evaluating the correctness of the composed model with respect to the given requirement specifications. A verified composed model guarantees the satisfaction of its objectives and required constraints through the consistent structure and coherent behavior of the composites. We believe these key contributions presented in this paper will assist academics and industrialists in the modeling, simulation, and analysis of real-time systems and further aid in complex engineering designs and implementations.

In future, we intend to explore the model composition and reuse of heterogeneous formalisms for conceptual modeling, multiple targeted executable modeling specifications and integration of broader range of verification tools and formal approaches for the composability verification.

APPENDIX

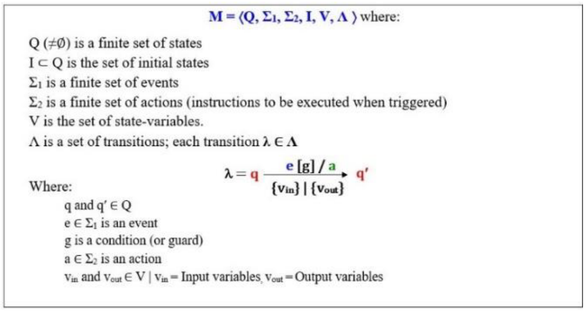
A. EXTENDED FINITE STATE MACHINES

The idea of EFSM was ignited by Cheng & Krishnakumar in 1993 [50]. EFSM is different from conventional finite state machine. As opposed to a conventional FSM, an EFSM model has complex transitions. "A transition λ is said to occur if the system is in a state **q**, an event **e** occurs, and the guard **g** is satisfied, then action **a** will be executed and the system will transit to the next state **q**′. An event is uniquely identified by an event name, has a set of parameters where each parameter is of primitive data-types, and is either of the type send or receive. A send event is an outgoing event whereas a receive event is an event expected from other EFSMs. During the firing of transition $\boldsymbol{\lambda} \in \boldsymbol{\Lambda}$ the state-variables $\{\mathbf{v_{in}}\}\in\mathbf{V}$ are used as input and the state variables $\{\mathbf{v_{out}}\}\in\mathbf{V}$ are used as output. This means that values of input variables are read when λ transition is being fired and is given as input to the action **a**, whereas the out variables are updated after the transition is fired by the action **a**. The state-variables could be of primitive types (e.g., *Boolean, Integer, String) and* or *complex types (e.g., list, set, and map)*. A complex type is a combination of basic types, defined in form of tuples''. Formally, an EFSM is defined by the tuple shown in Table 14.

B. ANYLOGIC EXECUTABLE ENVIRONMENT

Anylogic Simulation Software [24] provides a user-friendly executable environment with an efficient simulation engine for quickly creating and simulating models of complex systems. Anylogic provides a Java-based development environment and a set of multipurpose component libraries, which all

TABLE 14. Formal definition of extended finite state machine.



$M = \langle Q, \Sigma_1, \Sigma_2, I, V, \Lambda \rangle$ where:

Q ($\neq\emptyset$) is a finite set of states
$I \subset Q$ is the set of initial states
Σ_1 is a finite set of events
Σ_2 is a finite set of actions (instructions to be executed when triggered)
V is the set of state-variables.
Λ is a set of transitions; each transition $\lambda \in \Lambda$

$$\lambda = q \xrightarrow[\{v_{in}\} \mid \{v_{out}\}]{e\,[g]\,/\,a} q'$$

Where:
q and $q' \in Q$
$e \in \Sigma_1$ is an event
g is a condition (or guard)
$a \in \Sigma_2$ is an action
v_{in} and $v_{out} \in V \mid v_{in}$ = Input variables, v_{out} = Output variables

together help to speed up the modeling process. We propose to use the built-in libraries to create executable EFSM models and use the Anylogic Simulation engine, animation tool and graph visualization features to run and display simulation results, using different experiment configurations.

C. MODEL CHECKING

Model checking is used for the formal verification of our composed model. It is defined as: ''Model checking is an automated technique that, given a finite-state model of a system and a formal property, systematically checks whether this property holds for that model'' [51].

In formal logic, ''model checking designates the problem of determining whether a formula or a correctness property ϕ defined using Linear Temporal Logic (LTL), Computational Tree Logic (CTL) or similar property specification formalism, evaluates to true or false in an interpretation of a system **K**, written as $\mathbf{K} \models \boldsymbol{\phi}$'' [27]. Readers are referred to [27], [51] for the details of LTL & CTL constructs and their use. In this paper we use PAT (Process Analysis Toolkit) model checker. PAT is designed to analyze system models using various model checking techniques [46].

REFERENCES

[1] R. Fujimoto, C. Bock, W. Chen, E. Page, and J. H. Panchal, *Research Challenges in Modeling and Simulation for Engineering Complex Systems*. New York, NY, USA: Springer, 2017.

[2] R. B. Northrop, *Introduction to Complexity and Complex Systems*. Boca Raton, FL, USA: CRC Press, 2014.

[3] Y. Bar-Yam, ''General features of complex systems,'' in *Encyclopedia of Life Support Systems*. Oxford, U.K.: UNESCO, Eolss Publishers, 2002, p. 1.

[4] K. Popovici and P. Mosterman, *Real-Time Simulation Technologies: Principles, Methodologies, and Applications*. Boca Raton, FL, USA: CRC Press, 2013.

[5] L. Aceto, A. Ingólfsdóttir, K. G. Larsen, and J. Srba, *Reactive Systems: Modelling, Specification and Verification*. New York, NY, USA: Cambridge Univ. Press, 2007.

[6] E.-R. Olderog and H. Dierks, *Real-Time Systems: Formal Specification and Automatic Verification*. Cambridge, U.K.: Cambridge Univ. Press, 2008.

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

[8] O. Balci *et al.*, ''Model reuse, composition, and adaptation,'' in *Research Challenges in Modeling and Simulation for Engineering Complex Systems*, R. Fujimoto, C. Bock, W. Chen, E. Page, and J. Panchal, Eds. Cham, Switzerland: Springer, 2017, pp. 87–115.

[9] I. Mahmood, ''A verification framework for component based modeling and simulation: 'Putting the pieces together,''' Ph.D. dissertation, Dept. Softw. Comput. Syst., KTH Roy. Inst. Technol., Stockholm, Sweden, 2013.

[10] V. S. Alagar and K. Periyasamy, *Specification of Software Systems*. London, U.K.: Springer, 2011.

[11] H. S. Sarjoughian, ''Model composability,'' in *Proc. Winter Simulation Conf.*, Dec. 2006, pp. 149–158.

[12] S. J. E. Taylor *et al.*, ''Grand challenges for modeling and simulation: Simulation everywhere—From cyberinfrastructure to clouds to citizens,'' *Simulation*, vol. 19, no. 7, pp. 648–665, Jul. 2015.

[13] M. D. Petty and E. W. Weisel, ''A composability lexicon,'' in *Proc. Simulation Interoperability Workshop*, 2003, pp. 181–187.

[14] M. D. Petty, ''Verification, validation, and accreditation,'' in *Modeling and Simulation Fundamentals: Theoretical Underpinnings and Practical Domains*. Hoboken, NJ, USA: Wiley, 2010.

[15] O. Balci, J. D. Arthur, and W. F. Ormsby, ''Achieving reusability and composability with a simulation conceptual model,'' *J. Simul.*, vol. 5, no. 3, pp. 157–165, 2011.

[16] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi, ''Verifying dynamic semantic composability of BOM-based composed models using colored petri nets,'' in *Proc. ACM/IEEE/SCS 26th Workshop Princ. Adv. Distrib. Simul.*, Jul. 2012, pp. 250–257.

[17] T. Mens and P. Van Gorp, ''A taxonomy of model transformation,'' *Electron. Notes Theor. Comput. Sci.*, vol. 152, pp. 125–142, Mar. 2006.

[18] M. Brambilla, J. Cabot, and M. Wimmer, ''Model-driven software engineering in practice,'' *Synth. Lectures Softw. Eng.*, vol. 3, no. 1, pp. 1–182, 2017.

[19] S. Robinson, R. Brooks, K. Kotiadis, and D.-J. Van Der Zee, *Conceptual Modeling for Discrete-Event Simulation*. Boca Raton, FL, USA: CRC Press, 2011.

[20] K. Androutsopoulos, D. Clark, M. Harman, R. M. Hierons, Z. Li, and L. Tratt, ''Amorphous slicing of extended finite state machines,'' *IEEE Trans. Softw. Eng.*, vol. 39, no. 7, pp. 892–909, Jul. 2013.

[21] C. A. R. Hoare, ''Communicating sequential processes,'' *Commun. ACM*, vol. 26, no. 1, pp. 100–106, 1983.

[22] C. A. R. Hoare, ''Communicating sequential processes,'' *Commun. ACM*, vol. 21, no. 8, pp. 666–678, 1978.

[23] *PAT: Process Analysis Toolkit*. Accessed: Aug. 20, 2018. [Online]. Available: http://pat.comp.nus.edu.sg/

[24] Anylogic. (2019). *Anylogic Simulation Platform*. Accessed: Aug. 20, 2008. [Online]. Available: https://www.anylogic.com/

[25] B. P. Zeigler, T. G. Kim, and H. Praehofer, *Theory of Modeling and Simulation*, 2nd ed. Orlando, FL, USA: Academic, 2000.

[26] J. Peterson, *Petri Net Theory and the Modeling of Systems*. Upper Saddle River, NJ, USA: Prentice-Hall, 1981.

[27] N. Navet and S. Merz, *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, 1st ed. Hoboken, NJ, USA: Wiley, 2008.

[28] R. Alur and D. L. Dill, ''A theory of timed automata,'' *Theor. Comput. Sci.*, vol. 126, no. 2, pp. 183–235, Apr. 1994.

[29] R. Alur, *Principles of Cyber-Physical Systems*. Cambridge, MA, USA: MIT Press, 2015.

[30] C. Ptolemaeus, *System Design, Modeling, and Simulation Using Ptolemy II*, vol. 1. Berkeley, CA, USA: Ptolemy.org, 2014.

[31] M. Panarotto, J. Wall, M. Bertoni, T. Larsson, and P. Jonsson, ''Value-driven simulation: Thinking together through simulation in early engineering design,'' in *Proc. 21st ICED*, Vancouver, BC, Canada, vol. 4, 2017, pp. 513–522.

[32] J. Johannes, ''ATL use case-model driven performance engineering: From UML/SPT to AnyLogic,'' Book ATL, TU Dresden, Dresden, Germany, 2008.

[33] P. Mohagheghi, W. Gilani, A. Stefanescu, M. A. Fernandez, B. Nordmoen, and M. Fritzsche, ''Where does model-driven engineering help? Experiences from three industrial cases,'' *Softw. Syst. Model.*, vol. 12, no. 3, pp. 619–639, 2013.

[34] H. Saadawi, G. Wainer, and M. Moallemi, ''Principles of DEVS model verication for real-time embedded applications,'' in *Real-Time Simulation Technologies: Principles, Methodologies, and Applications*. Boca Raton, FL, USA: CRC Press, 2011, pp. 63–96.

[35] R. Gerth. (2007). *Concise Promela Reference*. Accessed: Aug. 20, 2008. [Online]. Available: https://spinroot.com/spin/Man/Manual.html

[36] P. Bouyer, S. Haddad, and P.-A. Reynier, ''Timed Petri nets and timed automata: On the discriminating power of zeno sequences,'' *Inf. Comput.*, vol. 206, no. 1, pp. 73–107, Jan. 2008.

[37] J. Liu *et al.*, "A calculus for hybrid CSP," in *Proc. Asian Symp. Program. Lang. Syst.*, 2010, pp. 1–15.

[38] L. Zou, N. Zhan, S. Wang, and M. Fränzle, "Formal verification of simulink/stateflow diagrams," in *Proc. Int. Symp. Automated Technol. Verification Anal.*, 2015, pp. 464–481.

[39] S. Gholami and H. S. Sarjoughian, "Modeling and verification of network-on-chip using constrained-DEVS," in *Proc. Symp. Theory Modeling Simulation*, Apr. 2017, p. 9.

[40] ACIMS. (2018). *DEVS-Suite Simulator 4.0.0*. Accessed: Aug. 20, 2018. [Online]. Available: https://acims.asu.edu/software/devs-suite/

[41] F. Moradi, R. Ayani, and I. Mahmood, "An agent-based environment for simulation model composition," in *Proc. 22nd Workshop Princ. Adv. Distrib. Simul.*, Jun. 2008, pp. 175–184.

[42] F. Moradi, R. Ayani, S. Mokarizadeh, G. H. A. Shahmirzadi, and G. Tan, "A rule-based approach to syntactic and semantic composition of BOMs," in *Proc. 11th IEEE Int. Symp. Distrib. Simulation Real-Time Appl.*, Oct. 2007, pp. 145–155.

[43] I. Mahmood, R. Ayani, V. Vlassov, and F. Moradi, "Statemachine matching in BOM based model composition," in *Proc. 13th IEEE/ACM Int. Symp. Distrib. Simulation Real Time Appl.*, Oct. 2009, pp. 136–143.

[44] A. Borshchev, *The Big Book of Simulation Modeling: Multimethod Modeling With Anylogic 6*. Chicago, IL, USA: Logic North America Chicago, 2013.

[45] S. J. Zhang and Y. Liu, "An automatic approach to model checking UML state machines," in *Proc. 4th Int. Conf. Secure Softw. Integr. Rel. Improvement Companion*, Jun. 2010, pp. 1–6.

[46] J. Sun, Y. Liu, and J. S. Dong, "Model checking CSP revisited: Introducing a process analysis toolkit," in *Leveraging Applications of Formal Methods, Verification and Validation*. Berlin, Germany: Springer, 2008, pp. 307–322.

[47] T.-W. Lin, S.-L. Hwang, and P. A. Green, "Effects of time-gap settings of adaptive cruise control (ACC) on driving performance and subjective acceptance in a bus driving simulator," *Saf. Sci.*, vol. 47, no. 5, pp. 620–625, 2009.

[48] I. A. Ntousakis, I. K. Nikolos, and M. Papageorgiou, "On microscopic modelling of adaptive cruise control systems," *Transp. Res. Procedia*, vol. 6, pp. 111–127, Jan. 2015.

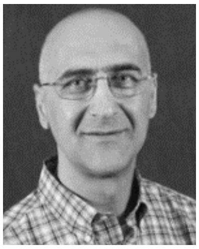
[49] L. Xiao *et al.*, "Realistic car-following models for microscopic simulation of adaptive and cooperative adaptive cruise control vehicles," *Transp. Res. Rec.*, vol. 2623, no. 1, pp. 1–9, Jan. 2017. doi: 10.3141/2623-01.

[50] K.-T. Cheng and A. S. Krishnakumar, "Automatic functional test generation using the extended finite state machine model," in *Proc. 30th ACM/IEEE Design Automat. Conf.*, Jun. 1993, pp. 86–91.

[51] C. Baier and J.-P. Katoen, *Principles of Model Checking* (Representation and Mind Series). Cambridge, MA, USA: MIT Press, 2008.

IMRAN MAHMOOD received the master's and Ph.D. degrees in computer systems from the School of Information and Communication Technology (ICT), KTH Royal Institute of Technology, Sweden, in 2007 and 2013, respectively. He is currently an Assistant Professor with the Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Pakistan. He is also serving as the Director of the Center for Research in Modeling and Simulation (CRIMSON). His current research interests include applied modeling, simulation, analysis, and formal verification of complex systems. He can be reached at imran.mahmood@seecs.edu.pk.

TAMEEN KAUSAR is currently pursuing the master's degree in information technology with the Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Pakistan. Her current research interests include model composability and modeling of complex systems. She can be reached attkausar.msit15seecs@seecs.edu.pk.



HESSAM S. SARJOUGHIAN is currently an Associate Professor of computer science and computer engineering with the School of Computing, Informatics, and Decision Systems Engineering (CIDSE), Arizona State University (ASU), Tempe, AZ, USA, and the Co-Director of the Arizona Center for Integrative Modeling and Simulation (ACIMS). His research interests include model theory, poly-formalism modeling, collaborative modeling, simulation for complexity science, and M&S frameworks/tools. He is the Director of the ASU Online Masters of Engineering in Modeling and Simulation Program. He can be contacted at hessam.Sarjoughian@asu.ed.

ASAD WAQAR MALIK received the Ph.D. degree in parallel and distributed simulation/systems from the National University of Science and Technology (NUST), Pakistan, in 2012, where he is currently an Assistant Professor with the Department of Computing (DOC), School of Electrical Engineering and Computer Science (SEECS). He is also a Senior Lecturer with the Department of Information Systems, Faculty of Computer Science and Information Technology, University of Malaya, Malaysia. His primary areas of interest include distributed simulation, cloud/fog computing, and the Internet of Things. He can be reached at asad.malik@seecs.edu.pk.

NAVEED RIAZ received the Ph.D. degree in computer engineering from the Graz University of Technology, Austria. He is with the Department of Computing, School of Electrical Engineering and Computer Science, National University of Sciences and Technology, Pakistan. His main research interests include software debugging, formal verification, and software engineering. He can be reached at naveed.riaz@seecs.edu.pk.