2020

# SAM-SoS: A stochastic software architecture modeling and verification approach for complex System-of-Systems

Ahmad Mohsin
*Edith Cowan University*

Naeem Khalid Janjua
*Edith Cowan University*

Syed M. S. Islam
*Edith Cowan University*

Muhammad Ali Babar



10.1109/ACCESS.2020.3025934
Mohsin, A., Janjua, N. K., Islam, S. M., & Babar, M. A. (2020). SAM-SoS: A Stochastic Software Architecture Modeling and Verification Approach for Complex System-of-Systems. *IEEE Access*, 8, 177580 - 177603. https://doi.org/10.1109/ACCESS.2020.3025934

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received August 12, 2020, accepted September 15, 2020, date of publication September 22, 2020, date of current version October 8, 2020.

*Digital Object Identifier 10.1109/ACCESS.2020.3025934*

# SAM-SoS: A Stochastic Software Architecture Modeling and Verification Approach for Complex System-of-Systems

**AHMAD MOHSIN[1], NAEEM KHALID JANJUA[1], (Member, IEEE), SYED M. S. ISLAM[1], (Member, IEEE), AND MUHAMMAD ALI BABAR[2]**

[1]School of Science, Edith Cowan University, Joondalup, WA 6027, Australia
[2]Centre for Research on Engineering Software Technologies (CREST), The University of Adelaide, Adelaide, SA 5005, Australia

Corresponding author: Naeem Khalid Janjua (n.janjua@ecu.edu.au)

This work was supported in part by the Edith Cowan University (ECU) Open Access Funding Scheme-2020 under Strategic Research Fund (SRF) and in part by Higher Degree by Research Scholarship (HDRS).

**ABSTRACT** A System-of-Systems (SoS) is a complex, dynamic system whose Constituent Systems (CSs) are not known precisely at design time, and the environment in which they operate is uncertain. SoS behavior is unpredictable due to underlying architectural characteristics such as autonomy and independence. Although the stochastic composition of CSs is vital to achieving SoS missions, their unknown behaviors and impact on system properties are unavoidable. Moreover, unknown conditions and volatility have significant effects on crucial Quality Attributes (QAs) such as performance, reliability and security. Hence, the structure and behavior of a SoS must be modeled and validated quantitatively to foresee any potential impact on the properties critical for achieving the missions. Current modeling approaches lack the essential syntax and semantics required to model and verify SoS behaviors at design time and cannot offer alternative design choices for better design decisions. Therefore, the majority of existing techniques fail to provide qualitative and quantitative verification of SoS architecture models. Consequently, we have proposed an approach to model and verify Non-Deterministic (ND) SoS in advance by extending the current algebraic notations for the formal models as a hybrid stochastic formalism to specify and reason architectural elements with the required semantics. A formal stochastic model is developed using a hybrid approach for architectural descriptions of SoS with behavioral constraints. Through a model-driven approach, stochastic models are then translated into PRISM using formal verification rules. The effectiveness of the approach has been tested with an end-to-end case study design of an emergency response SoS for dealing with a fire situation. Architectural analysis is conducted on the stochastic model, using various qualitative and quantitative measures for SoS missions. Experimental results reveal critical aspects of SoS architecture model that facilitate better achievement of missions and QAs with improved design, using the proposed approach.

**INDEX TERMS** Stochastic systems, system-of-systems, architecture modeling, quantitative verification, statistical model checking, system properties, formal modeling.

## NOMENCLATURE

### ABBREVIATIONS

| | |
|---|---|
| *ADLs* | Architecture Description Languages |
| *BTL* | Branching Time Logic |
| *CCS* | Calculus of Communicating Systems |
| *CPS* | Cyber- Physical System |
| *CS* | Constituent System |
| *CSL* | Continuous Stochastic Logic |
| *CSP* | Communicating Sequential Processes |
| *CTL* | Computation Tree Logic |
| *CTMC* | Continuous-Time Markov Chain |
| *EBNF* | Extended Backus–Naur form |
| *HSF* | Hybrid Stochastic Formalism |
| *IoT* | Internet of Things |
| *LTL* | Linear Temporal Logic |
| *MCCS* | Markov Chains for Concurrent Systems |
| *MDE* | Model-Driven Engineering |
| *ND* | Non-Deterministic |
| *NFRs* | Non-Functional Requirements |
| *PCTL* | Probabilistic Computation Tree Logic |
| *QAs* | Quality Attributes |

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

| | |
|---|---|
| *QKT* | Qualitative Reachability, with Known Thresholds |
| *QUT* | Quantitative Reachability with Unknown Thresholds |
| *SA* | Software Architecture |
| *SCCP* | Stochastic Concurrent Constraints Programming |
| *SoS* | System-of-Systems |
| *SPA* | Stochastic Process Algebras |

### SYMBOLS

| | |
|---|---|
| $\alpha_i$ | All individual named actions as $a, b.c, .., .$ of CSs |
| $\lambda_i$ | Random rate of actions $\alpha_i$ between CS |
| $\lambda_i$ | Random rate of actions $\alpha_i$ between CS |
| $\mathbb{Q}_{(i,j)}$ | Infinitesimal generator matrix for CTMCs |
| $\mathbb{R}^+$ | Set of real numbers |
| $\mathcal{P}$ | Probability or likelihood of reachability and QAs |
| $\mathcal{S}$ | Set of finite states with discrete or continuous time |
| $\models$ | Model satisfaction relation for states and paths |
| $\phi, \Psi, \Phi$ | State and path formulas with property $\Phi$ |
| $\pi$ | It is the path for exponential state transitions |
| $\pi^M$ | Set of paths in stochastic model $M$ |
| $ACT$ | Set of all the actions among CSs |
| $B\mathbf{p}$ | Binding protocol with ($Ports$, $Roles$) |
| $C_n$ | Set of concurrent constraints |
| $E_I$ | Set of exogenous interaction for CS |
| $E_{Total}(s)$ | Total exit rate of states in rate matrix |
| $G_B$ | CSs global behaviors |
| $G_p$ | Guard predicates used for constraining modules |
| $I$ | Set of interfaces for CSs |
| $L(f)$ | A labeling function for states: $s \rightarrow s'$ |
| $L_B$ | CSs local behaviors |
| $P, Q$ | Named processes used as CSs in HSF |
| $R$ | Transition rate matrix for states interactions in HSF |
| $s_{init}$ | Initial state at the start of CSs transitions |
| $T_r$ | Cartesian product of two transitions systems $Ts$ |
| $V$ | A collection of local and global variables |
| $V_{init}$ | Initial variables in stochastic modules |

## I. INTRODUCTION

A System-of-Systems (SoS) is a complex system that behaves in a stochastic manner resulting from the collaboration among various heterogeneous sub-systems known as Constituent Systems (CSs). The CSs are often distributed, exhibiting operational and managerial independence, but work together to achieve the SoS mission with the help of emergent behaviors which an individual CS alone could not accomplish [1], [2]. The emergent behavior, which is dynamic and results from unknown CSs interactions at runtime, makes overall SoS behavior stochastic [3]–[5]. Due to the independence and autonomy of SoS CSs, the administrators of the SoS have loose control over CSs, making it difficult to ensure the correct architectural design of SoS [1], [6], [7]. Modern mission-driven critical infrastructures designed as SoS provide services that are essential in daily life, including health, transportation, energy, emergency, and rescue services. If not designed properly, an SoS could fail, leading to the loss of human lives, disruption of core businesses and damages to economic growth [8]–[10]. Unlike traditional single systems whose components, structures, and behaviors are well known, it is challenging to design and implement SoS architecture since it is stochastic due to its unknown CSs, unpredictable behaviors, and continuous evolution [4], [11]. Therefore, the main focus of this study is to devise a unique formal modeling and verification approach for SoS architectures.

The Software Architecture (SA) modeling of complex software-intensive systems involves describing the functional features and performing structural analysis to determine potential defects and SA design issues. The SA design issues have a detrimental effect on Quality Attributes (QAs) such as performance, reliability, and security [12], [13]. A correctly designed SA for describing structure and behaviors coupled with constraints specification is crucial to software modeling and verification. Although SA provides specific modeling abstractions for the early prediction of defects and design issues, the current notations available for SoS modeling and verification lack the essential reasoning capabilities required to deal with the Non-Deterministic (ND) architecture. An SoS architecture is ND primarily because it is not known in advance whether the potential coalition of SoS (consisting of CSs that are autonomous and fully independent) can conform to the functionalities and QAs. Consequently, SoS structures, behaviors, and related QAs are not easy to predict and measure [11], [14]. Therefore, SA modeling and reasoning for such systems are challenging tasks that require strong mathematical foundations to specify stochastic behaviors and ND events in an unpredictable environment [15]. In this context, formal modeling and reasoning enable systems designers to specify and analyze SA models using a robust mathematical foundation.

Among various SA modeling tools, formal Architecture Description Languages (ADLs) are strong candidates for representing software systems architecture in the form of components (CSs), connectors (Mediators), and resulting configurations/coalitions [16]–[18]. The majority of these ADLs are based on core process algebras originating mainly from Calculus of Communicating Systems (CCS) [19] and Communicating Sequential Processes (CSP) [20] to model the SoS architecture [21]–[24]. However, they fail to deal with the stochastic behavior and dynamic nature of SoS [15]. On the other hand, some approaches try to model complex systems similar to these using Stochastic Process Algebras (SPA) [25]. Still, these formal ADLs individually based on process algebraic notations, i.e. CCS, CSP, SPA and related approaches [26] have certain limitations when it comes to modeling SoS [4], [11]. These limitations include: (a) vocabulary and reasoning capabilities to manage the

architectural characteristics of SoS (b) support for automated model verification [4] concerning missions and QAs and (c) inability to specify and reason dynamic stochastic behaviors and uncertainty of the SoS at the architectural level.

Verification of a complex system can be performed with statistical model-checking that enables various architectural analysis of system properties [27]. However, most of the current modeling approaches have a semantic mismatch making it difficult to perform automated quantitative verification analysis and reasoning. This requires models to retain semantic consistency and completeness during the transformation process, which needs to be addressed.

In this research work, we overcome these limitations with a unique approach based on Model-Driven Engineering (MDE) [28] that supports the stochastic architecture modeling of the SoS by using Markovian process algebra. This work makes several key contributions to this area of research. It is broadly categorized according to three aspects: (i) Syntax and semantics of SPA are extended with concurrent and stochastic composition operators into Hybrid Stochastic Formalism (HSF) as our proposed formalism. HSF brings features such as probabilistic choices, non-determinism and Stochastic Concurrent Constraints Programming (SCCP) constructs [29] to describe SoS architecture models. (iI) The resulting stochastic model specified using HSF is a Markovian model that supports Stochastic Model Checking (SMC). Formally founded mapping rules from proposed HSF to PRISM [30] are defined using formal semantics to perform automated verification analysis of the SoS model. (iii) Various system behavioral reachability and quantitative analysis of dynamic properties are performed with Continuous Stochastic Logic (CSL) on the transformed SoS model for the first time in PRISM using known and unknown bounds.

The proposed approach is validated using a case study of a Cyber-Physical System (CPS)-based SoS (CPSoS) for Real-Time Fire Monitoring and Emergency Response. The system has been modeled with the proposed HSF taking into account ND behavior and concurrent compositions. The probabilistic behavior has been tested for reachability employing approximate SMC. Steady-state and transient analysis are applied to predict QAs such as performance and reliability, using multiple scenarios to assess mission accomplishment qualitatively and quantitatively.

The rest of the paper is organized as follows. Section II describes the work related to our approach; background knowledge has been established in section III. The proposed approach has been elaborated next in section IV. The syntax and semantics of the proposed formalism have been extended in section V. Section VI provides an SoS architectural design with extended formalism. Section VII presents mapping rules from extended formalism into PRISM. Validation of the approach has been provided in section XIII with a case study implementation, including preliminary results discussions. A comparison of the proposed approach with existing works is performed in section IX. Finally, conclusions are drawn, and future work is discussed in section X.

## II. RELATED WORK

Work related to our proposed approach can be categorized into two broader bodies of knowledge: (i) Formal representation of complex systems architecture, especially (Formal Syntax and Semantics) and (ii) System architecture qualitative and quantitative analysis through model verification.

Over the past decade, there has been an emphasis on formal modeling of complex distributed systems to acquire insights into a system in terms of its architectural design and behaviour, and how it evolves over a period of time [31]. The most common and widely used formalisms for architecture description are categorized into: (i) Petri-nets, (ii) Queuing Networks, (iii) Z Notations (iv) Bi-graphs and (v) Process Algebras [32], [33]. Process algebra-based ADLs have increased in popularity and have established a place in the industry and academic research [16], [34], [35]. Architecture Description Interchange Language (ACME), enhanced with multiple formal representations, allows us to specify certain Non-Functional Requirements (NFRs)[1] with architecture structure and behavior using Wright and Rapide annotated properties [36], it has been extended for product line software system with aspect-oriented semantics. Architecture Analysis and Design Language (AADL) is a semi-formal notation for modeling systems structures and behaviors along with system properties [37]. Its syntax and semantics have been extended for the modeling of safety and hazard scenarios and detailed analysis using QaSten [38] approach for error modeling and verification.

For some time now, Electronics Architecture and Software Technology-ADL (EAST-ADL) has been used to model complex autonomous systems. Its application has been extended to modeling stochastic behaviors as it improves Clock Constraint Specification Language (CCSL) time-constrained semantics for probabilistic analysis [39]. ACME models are transformed into ALLOY[2] for performing verification, by integrating formal modeling notation of Wright into the Failures-Divergences Refinement (FDR) model checker; however, this approach has certain limitations in terms of QAs verification [40]. Cavalcante *et al.* [41] devised an approach for the verification of dynamic SA specified in $\pi-$ADL using the Plasma-Lab[3] SMC tool. For the verification of dynamic properties, DynBLTL was used, which is an extension of Bounded Linear Temporal Logic (BLTL) [42]. However, these model-checking processes cannot verify stochastic models and face the problem of state-space exploration [43].

In their work in [44] extended the Behavior Interaction Priority (BIP) [45] formalism to the stochastic formalism of Stochastic BIP (SBIP) [46] based on timed automata for enriched compositional modeling. They used Probabilistic BLTL (PBLTL)-based SMC algorithms to verify the properties of SBIP models, focusing particularly on performance evaluation. However, there was no explicit description of the

[1] NFRs are QAs such as performance, reliability, and security.
[2] https://alloytools.org/
[3] https://project.inria.fr/plasma-lab/

system threshold and the non-functional properties of the system. Song *et al.* [47] extended Monterey Phoenix (MP) formal modeling language with probabilistic automata for modeling software systems, using a model-checking tool based on Process Analysis Toolkit (PAT) to verify system behavior and quantitative evaluation. This approach used dead-lock checking and reachability analysis using Event-trace Linear Temporal Logic (LTL) based algorithms.

The aforementioned formalisms embedded into ADLs cannot be used to model and analyze SoS architectures as, predominantly, the formalism and vocabulary used in these approaches deal only with deterministic systems whose components, behaviors, and operating environments are already known to the system designers [3].

In their research work on SoS, Arnold *et al.* used UPDM/SysML profiles for modeling CSs as functional mock-up units, using Contract Specification Language to define the constraints on input and output [48]. The Plasma-Lab model-checking tool was used to perform SMC on stated execution traces of SoS. However, the underlying formalism used here is unable to reason about ND behaviors. Bozzano *et al.* [49] employed probabilistic model checkers with Compass Modeling Language (CML), a semi-formal SoS modeling language to perform safety verification. Sosadl [21] is a formal modeling language that specifies the deterministic structure and behaviors of SoS architecture. It is able to describe static architecture at abstract levels with intentional compositions. However, neither Sosadl nor CML support rigorous stochastic reasoning and cannot support qualitative and quantitative verification of behaviors and associated QAs. Moreover, they are unable to reason uncertainty concerning unknown CSs and their interactions.

Our approach is unique as it creates a modeling specification to describe SoS stochastic behaviors and dynamic structures in terms of runtime using rich syntax and semantics. The formally founded stochastic SoS models enable the formal verification and validation of system properties with steady-state and transient analysis. We try to enhance the SoS CS exogenous contractual behaviors by integrating CCS and CSP into the SPA as Markov models. Similarly, we use state-of-the-art SMC tool with unique stochastic model-checking algorithms for verifying dynamic emergent behavior. The proposed modeling approach enables SoS designers to verify qualitatively and quantitatively missions and goals from runtime perspectives at the design stage, taking into account the core architectural aspects of the SoS.

## III. BACKGROUND

### A. THE COMPLEX SYSTEM ORGANIZATION

SoS is a special type of complex system with increased complexity when implemented on a larger scale [50], [51] accompanying the physical components and information systems capabilities ranging from cybernetics-multi agents to computational biological systems [52]. Considering the types of SoS; i.e. collaborative and Virtual SoS, CSs are fully independent, geographically dispersed, and become part of SoS with partial contracts to achieve global missions by forming stochastic emergent behaviors dynamically [53]–[55]. On the other hand, centrally operated and managed SoS has non-stochastic behavior.

The architectural characteristics and types of SoS designed based on the level of autonomy, play a significant role in determining dynamic stochastic behaviors. This has been detailed in our research on SoS dynamic architecture modeling [7], [15].

Figure 1 depicts the complex nature of a SoS, evolving over a period of time, $T$ as coalitions of independent and autonomous CSs, collaborating to achieve a global mission in an uncertain environment. The uncertainty and the continuous evolution of SoS increase its complexity, impacting QAs such as performance and reliability that are critical to the fulfillment of the mission. A concrete example of such a SoS is an Emergency Response system comprising many heterogeneous and independent Internet of Things (IoT) (fire monitoring sensors, drones, police and ambulance services) as CSs, collaborating to achieve specific missions in the event of natural disasters or calamities (floods, hurricanes, and wild/bush fires incidents). However, the success of the mission and conformance of QAs in resulting coalitions is uncertain since CSs which are unknown may fail or mission may be compromised due to the unpredictability of CSs collaborations. Therefore, a SoS must be designed carefully early in its life cycle so that it can deal with the underlying SOS architectural complexity and minimize design bottlenecks.

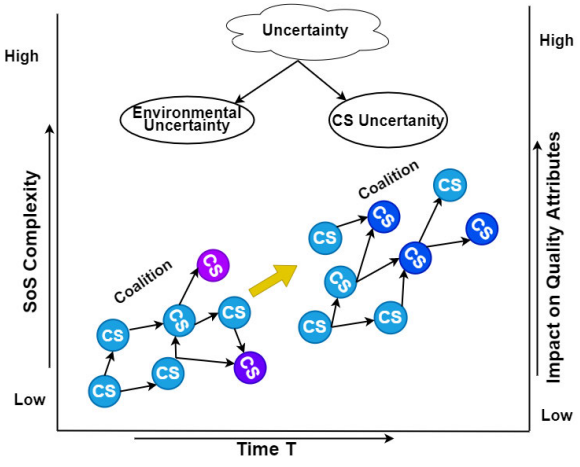



**FIGURE 1.** SoS architectural complexity and impact on system properties.

*Definition:* An SoS is a complex system $M$ and can be defined as tuple $M = < CS, I, C_H, G_B, R_{IB} >$ where:

- $CS = CS_i (i = 1 \ldots n)$, $n$ is a collection of independent systems as tuple in the form $CS_i = \langle L_B, E_I \rangle$ with certain time scales of $T \in \mathbb{R}^+ \geq 0$ as continuous time or discrete time starting from $t_0, \ldots, t_n$.

  - $L_B$: Individual local behavior of $CS_i$ that becomes part of a larger SoS.
  - $E_I$: It is a set of exogenous interaction for $CS_i \times CS_j$ that it provides to external environment and interacting CSs for collaboration.
- $I = (I_j)$ where $j = 1, \ldots, m$ and $I$ is a set of interfaces for interactive transition relations $(IR) = S_i \times S_j$ among independent CSs.
- $C_H : IO_{CH} \rightarrow I, O$, mediation for $IR$ as input and output channel with order sequence or in parallel communication.
- $G_B$: Global behavior are formed with the interaction of independent CSs $L_B$ as a result of $S_i \times S_j$interactions.
- $R_{IB}$: All the $E_I$ interactions and $L_B$ are essentially random generating $G_B$ stochastically.

In this definition, a SoS $M$ with architectural elements is a non-linear system, integrated with CSs such as IoT and CPS in social-technical and scientific contexts [10], [56], [57]. The architectural design of such complex systems needs to be described stochastically with underlying reasoning capabilities to avoid failures.

### B. STOCHASTIC SYSTEMS

SoS is a stochastic system that exhibits random concurrent actions where the CSs interactions lead to the probabilistic distributions. The CSs interactions are uncertain for future state reachability and primarily exhibit the properties of a Markov process. A stochastic process is a collection of random variables at time $T$ with each $t$ as: $(X_1, X_2, \ldots, X_n)$ with a function of: $t_0 < t_1 < t_2.. < t_n, \ldots, < t_{n+1}$ in the form $\{X_1(t_1), X_2(t_2), \ldots, X_n(t_n) \ldots X(t_{n+1})\}$ and collectively represented as: $\{X(t)\}_t \in T$.

A stochastic process is discrete if $\{X(t) \in T\}$ is observable at distinct points $T \in N^+$ or it is continuous if $T \in [0, \infty)$. In this research paper we selected the continuous time Markov process for modeling stochastic SoS.

*Definition Markovian Process:* A system acts as a Markovian process with a series of random variables if system states have a probability distribution as: $\mathcal{P}(X(t_1), X(t_2), \ldots, X(t_n)) = \mathcal{P}(X_{(tn+1)} \mid X(t_i)), t > 0$. In a stochastic process, the next state of a system can be described from current states of the system going from time $t$ to time $t + 1$ which is $\Delta t$ with time homogeneity property with transition from $i$ to $j$ we get:

$$\mathcal{P}_{(i,j)} = \mathcal{P}(X(\Delta t)) = j \mid X(t) = i \tag{1}$$

*Definition Memory-less Property:* At a given time $t_n$ the state $x^n$ of the system with probability $\mathcal{P}$ is independent of all previous states and dependent only on the recent one; i.e. $x^{n-1}$ at time $t_{n-1}$. This leads to a stochastic process exhibiting Markov property of being memory-less. Formally we can define this as:

$$(\mathcal{P}(Xt_n)(= x^n X(t_{n-1})) = x^{n-1} \Leftrightarrow \mathcal{P}(s \rightarrow s') = \left[s_{t+1} = s' | s_t = \mathcal{S}\right])$$

where:

- $\mathcal{S}$ is a set of finite states with discrete time or continuous time.
- $\mathcal{P}$ is a probability of moving from state $s$ to $s'$. With its memory-less property, the system's behavior can be predicted with the current state excluding the past states.

#### 1) STOCHASTIC INTERACTIVE FORMALISM

For modeling stochastic systems, Stochastic Interactive Formalism (SIF) originating from SPAs is the most suitable formalism for modeling and reasoning about probabilistic behavior and non-determinism [25], [58]. A SIF-based formalism leads to the formation of Markov labeled actions. These labeled actions with transition probability are represented as a set of actions:

$ACT = \{< a, b > U < F >\}$ with $a, b \in ACT$ as observable actions and $F$ unobservable actions. Observable actions are external actions of CSs through which a CS interacts with other CSs to achieve its objectives, for example, using public interfaces to send and receive messages. On the other hand, unobservable actions are internal control events of a CS through which core actions are managed. Examples are a CS reading and writing of data internally; such actions are usually private and concealed from other CSs. With a finite distribution of states $\mathcal{S}$ we obtain a distribution function as $f(x)$: $\mathcal{S} \rightarrow$ [0, 1]. As it is a finite set of states, it yields to $\sum s \in \mathcal{S}\, f(x)(\mathcal{S}) = 1$ *iff* $f(x) = 1$. This produces a Markovian distribution of $M_{Dist}(S)$ describing probabilistic distributions over states $\mathcal{S}$ of a system.

*Definition:* A stochastic formalism is a tuple $M = < \mathcal{S}, s_0, A, \mathcal{P}(A) >$ where:

- $\mathcal{S}$ is a finite set of states.
- $s_0 \in \mathcal{S}$ is the initial state.
- $A$ is a set of actions, such that $A \in ACT$ and it is represented as $ACT \rightarrow \mathcal{S} \times A \times M_{Dist}(\mathcal{S})$ forming a transition relation.
- $\mathcal{P}(A)$ is a probabilistic transition relation in $M$ as: $\mathcal{P}$ $\mathcal{S} \xrightarrow{(\alpha,\lambda)} \mathcal{S}'$ with $\lambda$ being a general probabilistic random rate and $\mathcal{P}$ is the probability that the transition of states will occur.

#### 2) LABELED TRANSITION SYSTEMS FOR CONCURRENT PROCESSES

A stochastic system is essentially a concurrent process $P_1 \parallel P_2, \ldots, \parallel P_n$ that forms Labeled Transition Systems (LTS), the behavior of which depends on the interactions of the processes and the environment as: $s \xrightarrow{(\alpha,\lambda)} s'$ and forms a transition relation $T_r$. By generalizing transition relation with $\mathcal{P}(A)$ we get:

$$T(r) \Rightarrow \mathcal{P}(A) \mid s \rightarrow s' = \sum_{A \in ACT} \{\lambda | S \xrightarrow{(A,\lambda)} S'\} \tag{2}$$

Here $\mathcal{P}(A)$ represents rate of action $\lambda$ for every $a \subseteq A$ representing transition probability from $s$ to $s'$ and $T_r$ is the transition relation over states.

*States Transitions and Paths:* The state transitions of components performing actions with certain timed rates can be traced on a particular path. Based on the execution traces of states, an infinite path is an infinite set of traces as:

$$Path(T_r) = \{(s_0, a_0, t_0), (s_1, a_1, t_1), \ldots\} \text{ so we obtain:}$$
$$\pi = s_0 \xrightarrow{(a_0,t_0)} s'_1, s_1 \xrightarrow{(a_1,t_1)} s'_2, \ldots, s_{n-1} \xrightarrow{(a_{n-1},t_{n-1})} s'_n$$

where $t \in T_r > 0$ and $a \in ACT$ and path is $\pi$ such that $\forall i > 0, R(s_i, s_{i+1}) > 0$. The finite path is a sequence of traces from $s_0 \rightarrow s_n$ with finite traces of execution with absorbing states. $s_n$ is the absorbing state for the system such that $\forall i$ and $T_r\ (s_n, s_{n+1}) > 0$. A particular path depending on the next state can be finite or infinite with the traces in a state space.

### 3) MARKOV CHAINS FOR CONCURRENT SYSTEMS

A stochastic system that has continuous/discrete state transitions in real-time is termed Markov Chains for Concurrent Systems (MCCS). Every transition in MCCS is associated with a rate or probability that shows the time it takes or the probability of moving from state $s$ to the next state $s_i$ leading to the exponential distributions of state space for system behavior.

*Definition:* At a given time $t$ a MCCS is a tuple of the form $N =< S, s_{init}, \lambda R, ACT, \mathcal{P}_{i,j}, \pi >$ where:

- $\mathcal{S}$ is a state space and $s_{init}$ is the initial state.
- $\lambda$ is the action rate or probability value for interaction among stochastic process.
- $R : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+ > 0$ is a transition rate matrix.
- $ACT$ is a set of actions as defined above.
- $\mathcal{P}_{i,j}$ is the probability $\mathcal{P}(s, s')$ of outgoing transitions from $s$ to $s'$.
- $\pi$ is the path for exponential state transitions of the system.

A path $\pi$ of MCCS is finite or infinite consisting of states $\pi(S_n, S_{n+1})$ for all $n \geq 0$. From MCCS we can derive Continuous-Time Markov Chain (CTMC) and Discrete-Time Markov Chains (DTMC) models substituting $\lambda$ with random rates (with $r$ for $\lambda$ and probability values) respectively. However, when these Markov models designed with process algebraic capabilities are coupled with LTS, more meaningful architectural descriptions for SoS can be specified.

A system can have reachable states if there is a finite path from $s$ to $s'$. Figure 2 shows CTMCs with 2 states in 3(a), 4 states 3(b) and 3 states in 3(c) respectively with their respective paths. Here, states shift from the current state to the next state with actions $ACT = \{a, b, c, d\}$ and the action rate is $\lambda$. The path for CTMC exhibits the race condition between the processes with origin state $s$ and successor state $s'$ and the rate is $R(s, s') > 0$. The probability of moving from state $s$ to $s'$ in time $t$ is defined as $1 - e^{-R(s,s').t}$ for time spent in each states, and the movement from s to s' in a single transition is called exit rate E(s): $\sum_{s \in S} R(s, s')$.

Throughout this paper, we use Markovian process formalism but extend and constrain it so that SoS can have specification reasoning capabilities for architectural modeling.

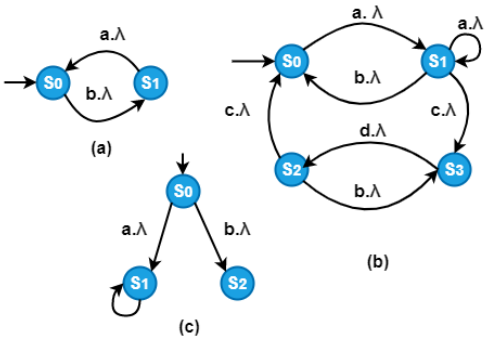



**FIGURE 2.** MCCS of (CTMCs and DTMCs) with actions rates and probabilities.

## C. TEMPORAL LOGIC AND VERIFICATION

A stochastic model $M$ with the characteristics of a Markov model can be verified against certain properties with temporal logic based on assumptions (systems behavior) and guarantees (properties of the system behavior) to be conformed [59], [60].

### 1) TEMPORAL LOGIC

*Definition:* Temporal logic is defined on the LTS with tuple $T_l =< M, AP, L(f) >$ where:

- $M$ is stochastic model to be verified.
- $AP$ is a proposition alphabet or a combination of atomic propositions ($APs$).
- $L(f)$: $S \rightarrow 2^{AP}$ $L(f)$ is a labeling function which attaches labels to the states.

Here $(AP, L(f))$ are used for specifying properties and testing whether $M$ satisfies certain properties as $(M \models \Phi)$ with $\models$ being the satisfaction relation over logical proposition $\Phi$. Various Boolean logic operators $(\wedge, \vee, \neg, \rightarrow, \leftrightarrow)$ are used for constructing propositional logic formulae using propositional logic. Furthermore, these can be used to check the states and paths of the stochastic model.

### 2) BRANCHING TIME LOGIC

A Computation Tree Logic (CTL)[4] formula uses Branching Time Logic (BTL) and can be represented with state formulas and path formulas with the following specifications:

$$\Phi ::= \text{APs} | \neg\Phi \mid \Phi \vee \Phi \mid \exists\phi \mid \forall\phi$$

Where $\exists\phi$ (there exists) represents a path of state(s) that fulfill $\phi$ and $\forall\phi$ (all paths) are satisfied by $\phi$. It does not hold with $\neg\Phi$ and $\Phi \vee \Phi$ meaning that either one of them satisfies the relation.

For path formulas we have $\phi ::= X\Phi \mid \Phi \cup \Phi$, $X$ is to ensure next states satisfy state properties. Here $\cup$ stands for until indicating $\Phi$ is true until the $\Phi$ is true. The probability for property $\phi$ and reachability of state $s$ from $s_0$ satisfying a path $\pi$ can be specified as:

$$\mathcal{P}(\phi) = \mathcal{P}(\pi \in Paths(Ts)(M, s_0) \mid \pi \models \phi)$$

[4]It is a form of Branching Time Logic (BTL) as compared to LTL for model verification of complex systems.

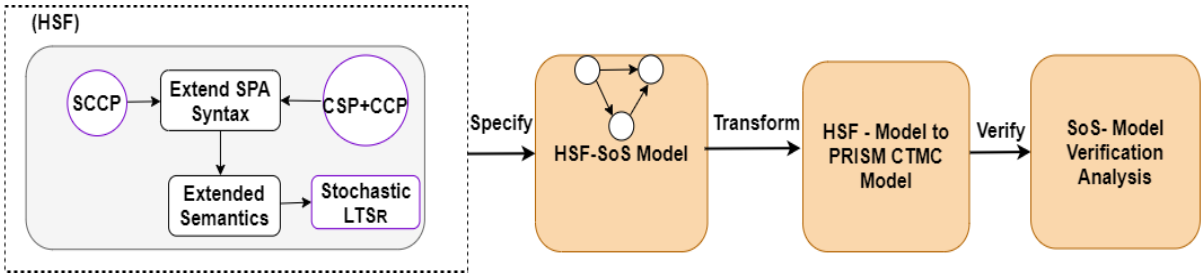



**FIGURE 3.** SAM-Sos architecture modeling and verification approach.

Using these base logic propositions, various steady-state and transient analysis can be performed on stochastic models based on BTL formulas as PCTL and CSL specifications [61].

### D. MODEL VERIFICATION

#### 1) PRISM MODELING SPECIFICATIONS

PRISM is a formal language integrated with a model checker, simulator, and system analysis sub-components. It supports symbolic state-based model verification for various stochastic models including CTMCs/ DTMCs, Markov Decision Process (MDP) and Probabilistic Time Automata (PTAs) [43]. These models are analyzed with property specification languages such as CSL / PCTL, LTL, and CTL for predicting system properties [62]. In our case, we use PRISM for stochastic model verification generated from our proposed formalism HSF. The PRISM modeling constructs consist of modules and variables equivalent to CSs as components and their behavior transition from state to state as concurrent compositions in the form of $CS_1 \parallel CS_2 \parallel CS_3, \ldots, \|\| CS_n$.

#### 2) FORMAL DEFINITION OF PRISM MODEL ELEMENTS

The mathematical foundation of PRISM is based on Alur's Reactive formalism [63], [64]. The semantics are defined as the compositional arrangement of modules in algebraic form for the interaction process.

*Definition:* The core elements of PRISM are stochastic concurrent processes which can be defined as tuple in the form $\mathcal{W} = \langle V,\ Vinit, G_p, T_R, C, \mathbb{M}\rangle$ where:

- V is a collection of local variables ($LV$) and global variables ($GV$).
- Initial variables are represented as $V_{init}$.
- $G_p$ is a set of guard predicates applied to guards for transitions to occur if predicates are met.
- $T_R$ is a Transition rate matrix $R$: $V\mathbf{x}\ V \rightarrow R_{STP\geqslant 0}$ that results due to updates in the variables.
- C represents commands $[]guards \rightarrow S_v$ resulted from $G_p$, $T_R$ and corresponding updates.
- $\mathbb{M}$ represents the set of modules **m** interacting stochastically in a concurrent manner. The module consists of local variables and commands.

The general syntax elements for PRISM language is provided here:

$$[ACT]\ guard \rightarrow S_v\text{: } Update_1 + , \ldots, + S_{v_n}\text{: } Update_n$$

The guard commands determine the system behavior with state transitions of local variables. When the command starts with an action represented by $ACT$ as a parallel composition of concurrent modules, the transition from the state is recorded as an update if the guard predicate is true for local variables. The stochastic information is presented with (stochastic value) $S_v$ that could be either probability $\mathcal{P}$ value if the model is DTMC/MDP, and random action rates $\lambda$ if it is CTMC.

## IV. STOCHASTIC ARCHITECTURE MODELING AND VERIFICATION APPROACH

This section presents an overview of the proposed approach for the modeling and verification of complex SoS architectures, as described in Section IIIA. Figure 3 depicts the proposed approach consisting of four core stages, starting from stochastic formalism specifications, leading to stochastic model development, and then transformations and model verification through model-checking. Each step involved in the proposed approach is described briefly below.

*Hybrid Stochastic Formalism:* At first, we integrate current process algebra with randomness, concurrency, synchronization, and concurrent constraints operators as a part of our proposed formalism as HSF. To provide syntax and semantics, we extend SPA with specific CCS, CSP, and CCP [65] operators into our proposed HSF, providing a compositional vocabulary for SoS architecture reasoning. HSF syntax and semantics are defined to establish formal foundations for modeling and analysis inspired by CML and Sosadl [21], [49]. This enables us to specify SoS architecture as a stochastic model to express concurrent compositions with probabilistic choices and non-determinism in CSs. A multi-labeled transition system with random actions $LTS_R$ enables the modeling formalism to generate probabilistic distributions of the stochastic model $M$ with execution state space $\mathcal{S}$ for the collaborative, dynamic behaviors of SoS. An EBNF[5] is generated from HSF syntax and semantics to orchestrate SoS architecture at an abstract level.

*HSF-SoS Model:* Secondly, we use stochastic architectural specifications using the HSF-driven EBNF, incorporating extended syntax and semantics for SoS. The structure and behavior of SoS are specified as CSs and mediators with ports

[5]HSF based syntax that generates the Extended Backus–Naur Form (EBNF) establishing process algebraic rules.

and roles by applying environment constraints to manage uncertainty. This enables us to generate SoS coalitions, which are stochastic and can be further reasoned for qualitative and quantitative analysis for rigorous evaluations of architectural models of SoS.

*Stochastic Model Transformation:* Model transformation is performed at this step. The HSF model can be treated as CTMC, which provides the abstraction for specifying SoS architectural elements and their interactions. However, the prediction of stochastic behaviors and their ability to achieve missions and conformance of QAs requires stochastic model verification. Therefore, the HSF CTMC model is transformed by proposing formal transformation rules in compliance with PRISM semantics. The formal rules allow automated transformation, enabling the analysis of the stochastic model from the HSF model. The one-to-one mapping is done by specifying formal rules for HSF and PRISM, which are compatible with both types of formal descriptions for modeling SoS.

*SoS Model Verification:* In the last step, a transformed stochastic model from HSF into the PRISM model-checker is used as CTMC model to verify the SoS stochastic architectural specifications. System properties are defined in CSL, in a unique way using known and unknown bounds for the evaluation of SoS missions and associated properties along with time-bounded logic specifications. Quantitative verification and predictions are made by applying relevant algorithms using CSL transient and steady-state analysis based on reachability and numerical computations [66].

## V. HYBRID STOCHASTIC FORMALISM FOR SoS

This section provides the extended syntax and semantics devised for formulating proposed HSF including abstract SoS architectural reasoning and coalition behavior. At an abstract level HSF for SoS is heterogeneous aggregation of concurrent processes $P$, actions $a \in ACT$, random rate of action $\lambda/(r)$ [6] $\in \mathbb{R}^+$ and Channels $\mathbb{C}$ forming a tuple as $< P, a, \lambda, \mathbb{C} >$. A process $P$ engages into action $a$ as $(a.P)$ with a probabilistic distribution over a time $T$ with action rate as $(a.\lambda).P$ which determines exponentially the duration or delay of actions. In order to constrain the interaction and deal with the uncertainty of exogenous interactions of CSs, we add concurrent constraint store operators annotated with random rates of CCP that enables the stochastic semantic to constrain the environment.

### A. SYNTAX

To compose SoS behavior and structure, we have used basic combinators defined in PA. The notion of a process or agent is retained with CSs where every CS is an independent process $P$ ranged over *A, B, C,..* as Names. Names are used for processes, and channels are used for communicating data between processes. We formally define syntax for HSF-SoS collaborating as concurrent processes:

$$(P ::= Skip \mid)(a.\lambda.P \mid)(a_x \rightarrow P \mid)(a_y \rightarrow Q \mid)(P_{(pchoice)}Q \mid)$$
$$(P \bowtie_r Q \mid)(P \rhd_u Q \mid)(\mathcal{A} \cong P)$$
$$(SCCP : P ::= Tell.\lambda(CStore) \mid ask.\lambda(Cstore) \rightarrow P)$$

The detailed description of each syntax element is as follows:

- *Skip*: This indicates that a process has been successfully terminated.
- $(a.\lambda).P$: This is an action prefix operator that presents a process $P$, performs an action $\alpha$ with activity rate $r$, and then again behaves as $P$. Various behavioral operators are used to determine the communication among CSs processes to interact and form Coalitions. This communication is categorized into sequential, parallel, and choices.
- $P_{(choice)}Q$: This shows probabilistic external and internal choices between $P$ and $Q$. With probability $\mathcal{P}$, it behaves as $P$ and as $Q$ with probability $\mathcal{P} - 1$ where probability choice $\mathcal{P} \in [0, 1]$ for choosing processes. The choice of processes is ND.
- $P \bowtie_{Lr} Q$: Parallel composition occurs for multiple events involving participating CSs. Actions could be both synchronous and asynchronous. For stochastic processes, parallelism is interleaved with a cooperation operator $\bowtie$ where the $r$ operator represents a list of actions that can be both hidden and silent.
- $P \rhd_u Q$: It runs as CS as $P$ if $P$ successfully terminates within a given time unit $u$; otherwise it behaves like $Q$.
- $Tell.\lambda(Cstore)$: This term tells the SoS environment about the new constraints imposed on a CS and added to the constraint store non-deterministically over a time interval index $i$ with a probabilistic distribution.
- $Ask.\lambda(Cstore)$: This term allows the participating CSs to derive certain information from constraints stores randomly over a time interval index with a probabilistic distribution.
- $\mathcal{A} \cong P$: $\mathcal{A}$ is a constant that assigns stochastic behavior for process $P$.

### B. SEMANTICS

The semantics of the above grammar can be established by a combination of axioms and inference rules for syntax operators. To build an LTS, the behavior of the system is derived via meaningful assertions. The inference rule $(P,\alpha,P') \rightarrow \mathcal{S} \times \mathcal{S}$ is a result of transitions $P \rightarrow P'$ with $s \rightarrow s' \in \mathcal{S}$ for process $P_i$ and $P'_j$ participating in actions $\alpha_i$.

*Definition Transition System with Rates ($LTS_R$):* The LTS for stochastic process is a tuple of the form: $< \sum_S, L_\alpha, T_{(r)} \rightarrow (f) >$ where, $\sum_S$ represents set of states $\mathcal{S}$ of the system, $L_\alpha$ is a set of label actions, $T_{(r)} \rightarrow \subseteq P \times P'$ is a transition relation and $f$ is a rate function of the form $(f)$:$\mathcal{S} \times L_\alpha \times \mathcal{S} \rightarrow \mathbb{R}^+$. This yields a probabilistic distribution for transitions of the system. Time for $T_{(r)}$ is $t \in R > 0$ and has

[6]Symbols $\lambda$ and $r$ are used here interchangeably to represent random action rates which lead to exponential distributions for system interactions.

function with value [0,1]. These processes can be represented with their action type and rate of action as:

$$\alpha(ActionType).P \rightarrow \alpha(ActionType),\ r(ActionRate).P'$$

The random rate $r$ leads to the exponential distribution of system behaviors over a period of time $t$ with probability being: $\sum f\ (P(\overrightarrow{a.r})Q)$. Now, by applying to the HSF syntax the general mechanism defined above and the general rule of premise and conclusion, we obtain abstract level transition rules by:

$$\frac{P_1\ \alpha_1 \longrightarrow P'_1, \ldots, P_n\ \alpha_n \longrightarrow P_n, \ldots, P'_m\ (\alpha_n.r)\ P'_m}{(\alpha.r).P \longrightarrow (\alpha.r).P'}$$

If the premise holds, then the conclusion also holds, and all the rules are symmetric. The transition relation is derived from the Cartesian product of two transitions systems say, $Ts_1$ and $Ts_2$ as: $(Ts_1 \parallel\ Ts_2)$. It works on rules if $s \rightarrow s'$ and conclusion pair of the Transition System (TS) obtained through a Cartesian product of the two transition systems leads to the formation of semantics as:

$$\begin{aligned}(Ts_1 \parallel Ts_2) &= ((\Omega_1) \times \Omega_2, ACT_1 \cup ACT_2)\\ &\Rightarrow (s_0(Ts_1) \times (s_1(Ts_2), L(f)))\end{aligned}$$

where $\Omega$ is the state space $\mathcal{S}$, $ACT$ is the set of actions, initial states are of the form $s_0$ and $s_1$ and $L(f)$ represents a labelling function. These principles of concurrent systems transitions provide baselines for the the formation of HSF semantics.

### 1) AXIOM RULES

Axiom rules are encapsulated in $LTS_R$, and are the possible representations of the terms defined in the aforementioned grammar. In the semantic rules, $TR$ signifies Transition Rule while $P, Q, P', Q'$ represent the process involved in actions $\alpha$ which are transformed with a condition as: if $P$ transforms with $(\alpha.r)$ to $P'$ then in return the transition relation of parallel processes $(P \parallel Q)\ P'$ is reached. Semantic operations based on premise and conclusion rules are as follows:

$$\textbf{Prefix} T_{r0(P)} : P \xrightarrow{(\alpha,r)} \langle P \rangle$$

$$\textbf{Probabilistic Choice} : (T_{r1(P)}) : \frac{P \xrightarrow{(\alpha,r)} P'}{\langle P(\delta)Q \rangle \xrightarrow{(\alpha,r)} \langle P'(\delta)Q \rangle}$$

$$(T_{r1(Q)}) : \frac{Q \xrightarrow{(\alpha,r)} Q'}{\langle P(\delta)Q \rangle \xrightarrow{(\alpha,r)} \langle Q'(\delta)Q \rangle}$$

$$\textbf{Parallelism} : (T_{r2(P,Q)})$$

$$: \frac{P \xrightarrow{(\alpha,r)} P'}{\langle P \bowtie_L Q \rangle \xrightarrow{(\alpha,r)} \langle P' \bowtie_L Q \rangle}((\alpha \notin L))$$

$$(T_{r2(Q)}) : \frac{Q \xrightarrow{(\alpha,r)} Q'}{\langle P \bowtie_L Q \rangle \xrightarrow{(\alpha,r)} \langle Q' \bowtie_L Q \rangle}((\alpha \notin L))$$

$$(T_{r2(P,Q)}) : \frac{P \xrightarrow{(\alpha,r)} P'}{\langle P \bowtie_L Q \rangle \xrightarrow{(\alpha,r)} \langle P' \bowtie_L Q \rangle}((\alpha \in L))$$

### 2) STOCHASTIC CONCURRENT CONSTRAINTS AXIOMS

The non-determinism of state transitions when communicating CSs and uncertainty can be managed with the application of stochastic concurrent constraints. CCP is used for the SoS model in order to constrain the interactions among CSs; it acts as a mediator to manage concurrent systems by adding new information about CSs by means of $Tell()$ and $Ask()$ operators. This forms a function $C(f)$:$C(Store) \rightarrow \mathbb{R}^+$ associating a rate for constraint store with real number:

$$(Tell.\lambda) : \frac{P[< C_n >] \xrightarrow{(\alpha,r)} P'[< Cn >]}{\langle P[< C_n >]Q \rangle \xrightarrow{(\alpha,r)} \langle P'[< C_n >]Q \rangle}$$

$$(\Rightarrow) \frac{P[Tell < C_n >] \xrightarrow{(\alpha,r)} P'[Tell < C_n >]}{\langle P[Tell < C_n >]Q \rangle \xrightarrow{(\alpha,r)} \langle P'[Tell < C_n >]Q \rangle}$$

where $< C_n >$ is the set of constraints in the constraint store that transforms $P$ to $P'$. The constraints are inferred by means of operator $Tell$ and $Ask$ with the association of randomness as $\lambda$:

$$(Ask.\lambda) : \frac{P[< C_n >] \xrightarrow{(\alpha,r)} P'}{\langle P[< C_n >]Q \rangle \xrightarrow{(\alpha,r)} \langle P', Q \rangle}$$

$$\Rightarrow \frac{P[when < C_n > do] \xrightarrow{(\alpha,r)} P'}{\langle P[when < C_n > do]Q \rangle \xrightarrow{(\alpha,r)} \langle P', Q \rangle}$$

### 3) INTERLEAVING COMPOSITION

Since a SoS is stochastic in nature, a stochastic process is a discrete continuous-time process of CMTC generated through LTS exponential probabilistic distribution of system state transitions. For every process, the overall rate at which actions are performed is termed $r.\alpha_i(P,Q)$ where $\alpha$ and $r$ represents action and rates respectively. From these semantic transition rules, a stochastic model of a SoS can be treated as a multi-labeled transition system leading to transition systems $Ts = T_{r1} \parallel T_{r2} \ldots \parallel T_{rn}$ interacting independent systems combined and generalized we obtain:

$$\left(\sum CS, A_{\alpha i}, \left\{\xrightarrow{(\alpha,r)} |(\alpha, r) \in ACT\right\}\right) \leftrightarrow \sum Ts \qquad (3)$$

where $\sum CS$ is the set of constituent systems, $A_{\alpha i} \in ACT$ is the set of activities / actions $(a.r)$ and the multi-transition relation is represented as $Ts$. Such systems behave in a ND fashion when an action is performed that affects certain interacting systems as shown in Figure 4. By expansion law for interleaving semantics and memory-less property, interleaving parallel composition in the form of $(P1 \| P2).\lambda$ states that reachability is the $\lambda$ delay time rate with exponential distributions. Suppose that we have two CSs as $P$ and $Q$ where $P$ has action $\alpha.\lambda$ and $Q$ has action $\beta.\lambda$, and these form a parallel composition, e.g. $(\alpha.\lambda \| \beta.\lambda) \cong (\alpha.\lambda, \beta.\lambda + \beta.\lambda, \alpha.\lambda)$. By means of an interleaving operator, such processes choose actions non-deterministically as shown in the transition system in the figure below.

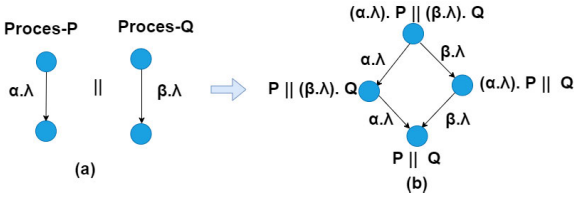



**FIGURE 4.** Interleaving semantics between two interacting systems.

*Definition Behavior Transitions:* Above $LTS_R$ leads to a Markov model with the behavior transitions of a CTMC in the form of a tuple $< \mathcal{S}, R, \mathcal{P}_i >$ with $\mathcal{S}$ being finite set of states, $R$ is the rate matrix with a function $R : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+ \geq 0$ and $\mathcal{P}_i$ is the probability distribution at the start $\mathcal{P}_i{:}\mathcal{S} \rightarrow [0, 1]$ such that it yields to $\sum_s \in S\mathcal{P}_i(\mathcal{S}) = 1$. Formally it can be represented as rate of moving from state $i$ to state $j$ in $T_s$ as $R(i, j) \rightarrow R(s, s')$. Then with the probability of moving from state $s$ to $s'$ at an exponential rate, $\lambda$ we get $R(s.s') = \lambda.\mathcal{P}(s, s')$. If there are many possible states $s$, then it will transition to next state $s'$ with the shortest time, known as a race condition. The exit rate of all outgoing transitions is denoted by $E(s)$ and this can be generalized as the total exit time from a particular state:

$$E_{Total}(s) = \sum_{s=s'} s \in S \quad R(s, s') \tag{4}$$

The transition of the form $R(s, s') = 0$ leads to an absorbing state in the state space with $E(s) = 0$. The probability of leaving non-absorbing state $s$ in time interval [0, t] of the time function $f(t)$ is exponentially distributed as: $\mathcal{P}(f(t)) = 1 - e^{-E(s).t}$. Similarly, the probability of moving from $s \rightarrow s'$ of non-absorbing state in time interval [0, t] is with E(s) from (4):

$$\mathcal{P}(f(t)) : (s \rightarrow s') = \frac{R(s, s')}{E(s)}.(1.e^{-E(s).t}) \tag{5}$$

*Definition States transitions over Continuous Time:* The dynamic behavior of system execution in its state space $\mathcal{S}$ can be traced and analyzed using a rate matrix $\mathbb{Q}$ depicting the rate of constant movement between states. Moving from state $i$ to $j$ depends on current states and it ignores past states that, essentially, are memory-less property of stochastic processes. For all $\mathcal{P}_{i,j}$ for $(i \neq j)$ and $(i = j)$ the formal notations for the infinitesimal generator matrix are:

$$\mathbb{Q} = \begin{cases} \lambda_i, \mathcal{P}_{i,j} & i \neq j \\ -\lambda & i = j \end{cases}$$

The transition diagonal matrix is in the form $\mathcal{P}[i, j]$ that yields $\sum_{J=i} R(i, j)$ with $1 \leq i \leq n$, then and CTMC transition matrix is formulated as:

$$\mathbb{Q} = (R - E(s)) \tag{6}$$

Using row column notation with diagonal elements $q_{i,j} = -\sum_{j=i} q_i$ and from (6) $\mathbb{Q}$ the matrix is formulated showing state transitions resulting from $T_s$ of (3):

$$\mathbb{Q}_{(i,j)} = \begin{bmatrix} -a_{11 \Rightarrow \lambda} & a_{12} & \cdots & a_{1n} \\ a_{21} & -a_{22 \Rightarrow \lambda} & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{m1} & a_{m2} & \cdots & -a_{mn} \Rightarrow \lambda \end{bmatrix} \tag{7}$$

where $a_1 a_2, ..a_n$ are individual state transitions of the form $s_0, s_1 \ldots s_n$ and a $\Rightarrow \lambda$ are the diagonal elements occupying exit rates E(s). The probability of moving to the next state $j$ is provided by the time unit $\Delta t$ from (7). The $\mathbb{Q}$ matrix allows system designers to compute a system's steady-state and probability of transient states with the help of vector $\mathbb{P}$ . $\mathbb{P}$ vector is used to indicate the probabilities of a system being in the initial state. It is defined formally as:

$$\mathbb{P} = \begin{bmatrix} s_1 \\ s_2 \\ \vdots \\ s_n \end{bmatrix} \tag{8}$$

Here, $s_1, \ldots, s_n$ indicate the probabilities of a system in state $s_i$ at time $t_i$. If there is no change in long-run steady-state then by applying limit it yields to $\mathcal{P}_i = \lim_{t \rightarrow \infty} \mathcal{P}_i(t)$ and $\mathcal{P}_i = 0$.

The system of linear equations is a product of $\mathbb{Q}$ matrix and vector $\mathbb{P}$ that enable system behavior to be explored as will be explained further in subsequent sections. By applying various algorithmic combinations to Markovian models, a range of analysis and predictions can be performed on complex SA using stochastic propositional logic.

### C. ABSTRACT ARCHITECTURE LEVEL REASONING SEMANTICS

Here, we provide the core of the semantics at the architectural level for CSs and mediators, as depicted in Figure 5. These semantics enable architectural level reasoning and establish the basis for performing further systems analysis.

#### 1) CONSTITUENT SYSTEM SEMANTICS

CSs interfaces are paired with $(Port, Role)$ association, with each port assigned a role by mediator. Here, port **P** describes the external, exogenous behavior of every CS in association with Role **R** defined by mediator M. So for every role of mediator M there is a port association that forms a binding protocol $(B_{\mathbf{p}})$ as:

$$(B_{\mathbf{p}} \parallel)(B_{\mathbf{p}} \parallel (\mathbf{R}_1, \mathbf{P}_1), \parallel (\mathbf{R}_2, \mathbf{P}_2) \ldots . \parallel (\mathbf{R}_n, \mathbf{P}_m))$$

$$(B_{\mathbf{p}}) = (\sum_{i=1}^{n,m} (\mathbf{R}, \mathbf{P}))$$

The behavior of the CSs is constrained by protocols in the form of contracts among ports for information exchange. The participating CSs can request and reveal certain information to the SoS environment at random rates to cope with the uncertainty. Here, the SoS has no control of the internal behavior of CSs, so it is important to agree on some contracts

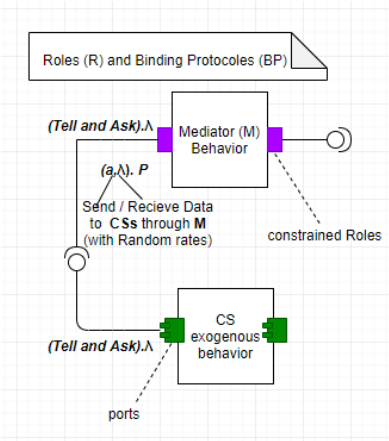



**FIGURE 5.** Mediator and CS semantics at architectural level.

through which architectural elements can collaborate and exchange information.

#### 2) MEDIATOR SEMANTICS

Similar to CSs which have certain interfaces and ports, Mediators have certain roles $\mathbf{R}$ and Bind Protocols $B_{\mathbf{p}}$ that help to coordinate with CS interfaces i.e. ports and this facilitates the accomplishment of tasks. The coordination of these events is done through $B_{\mathbf{p}}$ in a specific order. Hence, it is clear that interactions roles $\mathbf{R}$ and $B_{\mathbf{p}}$ are parallel actions enabling CSs to communicate. So we obtain:

$$B_{\mathbf{p}} \parallel \mathbf{R}_1 \parallel \mathbf{R}_2,\ldots,\parallel \mathbf{R}_n\ B_{\mathbf{p}} = \sum_{i=1}^{n} \mathbf{R}_i$$

Every role that the mediator defines for a system is assigned a relevant functionality that is to be achieved during system execution. From above equation, Joint protocol is a sum of all possible actions relevant to the specified roles of CSs.

### D. SoS COALITION BEHAVIOR

The coalition behavior with Algorithm 1 is formulated based on the stochastic $LTS_R$ semantics for the Markov model $M$. At a given time $t$, the architectural model $M$ for SoS forms a coalition generating a stochastic behavior, which is a result of local behaviors as: $\mathcal{C} =< L_B, (M) >$ where $L_B$ is local behavior in $M$ as defined in section IIIA along other architectural elements. Algorithm 1 takes as input the initial SoS coalition $\mathcal{C}$. It starts with an empty set of $G_B$ at line 1 and probabilistically adds up incrementally the local behaviors $L_B$ of each CS. State transition matrix $R$ at line 3 takes up the local behaviors and looks for transitions with interfaces $I$ for each architectural element in the coalition. From line (4-5), it determines binding protocol $B_{\mathbf{p}}$ through the ports and roles $(Port, Role)$ of interfaces described in section VC. The set of new transitions $T_r$ of labelled actions updating $R'$ is added with the probability distribution of identifying the source and target states using ports, roles over $IO_{CH}$ connections at lines (6-7).

The resulting behavior of the coalition is assigned to $G_B$ at line 10, which is a result of initial and new states transitions satisfying stochastic Concurrent Constraints (CCs). The last line ensures that stochastic behavior is a result of the local actions of CSs. Algorithm 1 constrains SoS concrete architecture coalition behavior generated from proposed HSF. It brings the architectural elements of SoS together to generate global behaviors with probabilistic distributions using underlying HSF semantics. The stochastic behavior is generated as a CTMC Model, which is used in PRISM for SoS model specification.

**Algorithm 1** Stochastic Model Behavior

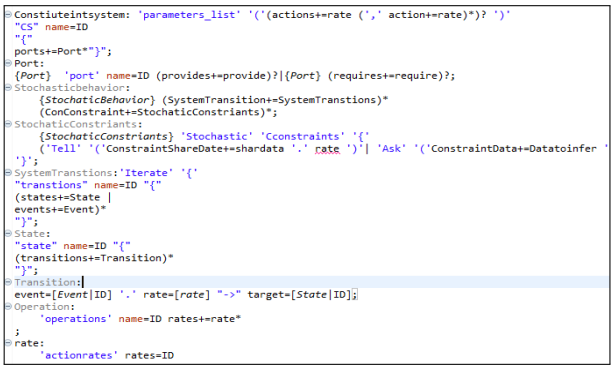
**Require:** $\mathcal{C} =< \sum_{i=1} L_B \parallel L_B, (M) >$
1: $G_B \leftarrow \emptyset$
2: **for all** $n \in N$ **do**
3: $(s, s') : R' \leftarrow \emptyset\ \{t \in R \mid \sum L_{Bn} = (S, s_{init}, R) \wedge I(t)\}$
4: **for all** $t \in R$ **do**
5: $B_{\mathbf{p}} \leftarrow \{(Port, Role) \in \sum B_{\mathbf{p}} \mid (Port(\mathbf{P}))=n \vee Role = n \wedge ip(\mathbf{P}) = I(t)\}$
6: **for all** $IO_{CH} \in B_{\mathbf{p}}$ **do**
7: $R' \leftarrow R' \cup \{S_i(t) \longmapsto \mathcal{P}(t) \mid B_{\mathbf{p}}.LabelActions(IO_{CH})\}$
8: **end for**
9: **end for**
10: $G_B \leftarrow G_B \cup (S, s_{init}, CCs(R' \backslash R) \cup (R'))$
11: **end for**
**Ensure:** $G_B = (L_{B1} \parallel L_{B2}, \ldots \parallel L_B) \forall n \in N$

### E. FROM HSF SEMANTICS TO DSL

The HSF is then integrated into EBNF to generate a stochastic Domain-Specific Language (DSL),[7] which is used for describing SoS architecture using the MDE approach. The meta-models are established using grammar rules to create a high-level DSL. For this purpose, an EMF[8]-based, Xtext[9] approach is used that allows us to parse, build and translate the internal code into an external DSL for architectural representation. A high-level DSL allows the system designers to describe the architectural elements of SoS with stochastic reasoning by emulating the underlying syntax and semantics within meta- models.

Figure 6 depicts the EBNF rules based on HSF semantics using Xtext for SoS CS behavior specification. The CS definition starts with its name, state, and ports declaration. The behavior comprises exogenous actions with random delay rates and stochastic constraints using tell and ask operators to deal with uncertainty. CS transition rules are defined with the set of states consisting of events with rates. This leads to generating probabilistic distributions of CSs when interacting with other CSs that have collective dynamic transitions.

[7]HSF based external (DSL) that enables abstract level SoS architectural representation.
[8]https://www.eclipse.org/modeling/emf/
[9]https://www.eclipse.org/Xtext/index.html



```
Constiuteintsystem: 'parameters_list' '('(actions+=rate (',' action+=rate)*)? ')'
 "CS" name=ID
 "{"
 ports+=Port*"}";
Port:
 {Port}  'port' name=ID (provides+=provide)?|{Port} (requires+=require)?;
Stochasticbehavior:
     {StochaticBehavior} (SystemTransition+=SystemTranstions)*
     (ConConstraint+=StochaticConstriants)*;
StochaticConstriants:
     {StochaticConstriants} 'Stochastic' 'Cconstraints' '{'
     ('Tell' '('ConstraintShareDate+=shardata '.' rate ')'| 'Ask' '('ConstraintData+=Datatoinfer '
 '}';
SystemTranstions:'Iterate' '{'
 "transtions" name=ID "{"
 (states+=State |
 events+=Event)*
 "}";
State:
 "state" name=ID "{"
 (transitions+=Transition)*
 "}";
Transition:
 event=[Event|ID] '.' rate=[rate] "->" target=[State|ID];
Operation:
     'operations' name=ID rates+=rate*
 ;
rate:
     'actionrates' rates=ID
```

**FIGURE 6.** Excerpt-CS EBNF rules for DSL based on HSF.

Similarly, grammar rules for mediator behaviors and abstract SoS architecture are defined to allow system architects to describe SoS architecture using simple architectural notation. This hides from the system designers the internal complexity of underlying formalism and provides flexibility to describe SoS models, which can be reused in the future. The generated DSL based on HSF provides a formal stochastic basis for the qualitative and quantitative verification of SoS architecture.

This section makes significant contributions to the body of knowledge regarding the modeling of SoS stochastic structures and behaviors. The underlying semantics of HSF, bring reasoning capabilities among CSs events with probabilistic choices, race conditions, and conditional synchronous compositions to generate model behaviors. The SCCP operators are used to deal with uncertainty in SoS architectural models at the interactions level.

## VI. CASE STUDY- EMERGENCY RESPONSE SYSTEM AS CPSOS USING HSF

Architecture modeling and verification by means of the proposed approach are conducted through a case study design of a mission-critical real-time Fire Monitoring and Emergency Response SoS (FM-ERSoS). It is a part of a smart city project that allows various departments and entities to collaborate, particularly when dealing with emergency situations. The emergency system is inspired by smart city projects to manage disaster situations by encompassing modern IoT and CPS technologies [67], [68] as CPSoS (consisting of various IoT-based CPS nodes and third party independent CSs). emergency response SoS, in collaboration with remotely distributed software-intensive systems, deals specifically with sudden fire eruptions in urban and rural areas with continuous monitoring.

Due to its architectural characteristics, FM-ERSoS exhibits ND behavior with exponential distribution comprising random action delays. Therefore, it is a challenge to design such a critical system that dynamically evolves to achieve missions [57], [69]. Essentially, it is a collection of various independent CSs as CPS nodes and embedded physical resources enriched with computing logic and interconnected and able to sense data from the environment and subsequently make a decision [70], [71]. Each CPS node is equipped with different sensors, connected through Wi-Fi Networks and Wireless Protocols (IEEE802.11 Wi-Fi and Zigbee). Various CPS nodes further collaborate through local gateways capable of performing the necessary processing.

### A. ARCHITECTURAL DESIGN CONSTITUENTS

Figure 7 (generated from UML based on FM-ERSoS metamodel) shows the high-level architectural view with architectural elements and flow of information for FM-ERSoS.

Following are the key architectural components of FM-ERSoS:

- Heat, Humidity and Wind-flow Sensors (CPSHWF-nodes).
- Flame and Smoke Monitoring (CPSFS-nodes).
- Local Control Station (LCS-nodes).
- Emergency Control Unit (ECU).
- Mediators with Wireless Sensor Networks (WSNs) for mediation among CS.

### B. STOCHASTIC ARCHITECTURAL SPECIFICATION

Specific to FM-ERSoS, Figure 8 shows a CS architectural specification with HSF-based semantics as outlined in Section V. CS, named CPSFS, is integrated locally with fire and smoke IoT sensors as a complete and independent system. This CPS node is responsible for predicting/detecting fire in the early stages, performing the necessary measurements, and transmitting these to the nearby CPS. This CPS sends fire data in real-time as sensors detect smoke and flames in the environment. Figure 7 depicts the coalition of FM-ERSoS, which generates behavior with different CSs through the constrained coordination of mediators.

In order to deal with unexpected wild-bushfire, the system consists of multiple CSs nodes working as CPSHWF, CPSFS that enable the detection of fire and send data in real-time to LCS. In response, the LCS nodes issue immediate emergency and warning alerts to the ECU. Starting with the CPSFS, as aforementioned has ports providing interfaces, namely send-fire-data and receive-fire-data, at random rates. The random actions with delays lead to the exponential distribution of system state transitions. The mediator named **M** as WSN coordinates fire monitoring data with specified roles of data communication using the CS port protocols. Here, the sending and receiving of fire data are achieved with corresponding CSs ports signatures and roles. The second mediator defined here coordinates with the fire monitoring nodes CPSFS and CPSHWF which continuously gets data from fire areas in order to monitor the real-time situation. To manage uncertainty, the mediators and CSs exchange exogenous external information through stochastic concurrent constraints via the Tell and Ask operators.

### C. FM-ERSoS MISSIONS AND QUALITY ATTRIBUTES

FM-ERSoS has two main missions namely $M_1$ and $M_2$. The first and by far the most important mission $M_1$ is to detect the fire events as soon as possible and send messages to ECU.

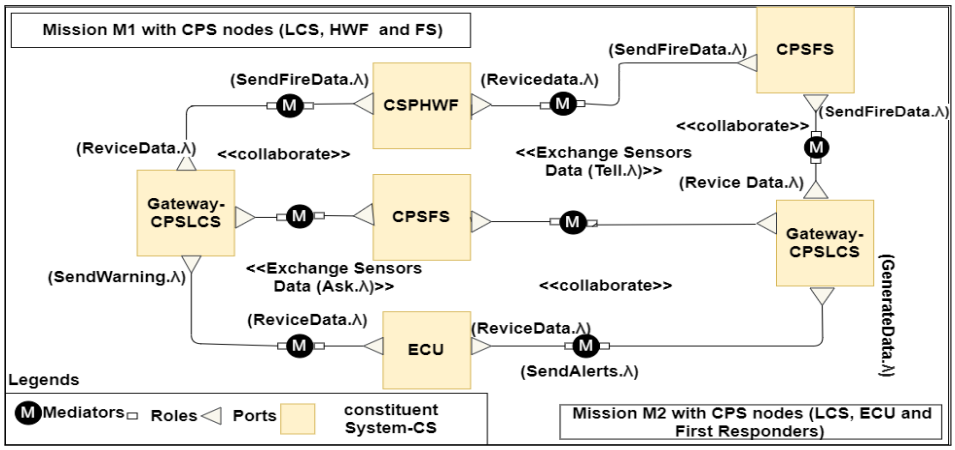



**FIGURE 7.** FMER-SoS concrete architecture with missions.

```
// Architectural Specification for Flame and Smoke detection and measurement CPS
CS CPSHWF parameter list (H_rate, WF_rate, l_rate) {
Port SendFiredata
CPS IoTnodes {
IoTnode1 Humiditysensor
IoTnode2 Windflowsensor
IoTnode3  GPS-sensor}
Iterate {
Hdata = Humidtysensor fettchdata.H_rate
WFdata = WindflowSensor.fetchdata.WF_rate
locationdata= GPS.get_coordinates()}
Stochastic CConstraints{
Tell.(sensorslocation.l_rate) through {locationdata}
Ask. (nearbynodes.l_rate) through {locationdata}
communicatdata {
Throughconnection Hdata send.H_rate
Throughconnection WFDdata send.WF_rate}}
```

**FIGURE 8.** Abstract CPSFS behavior description in HSF.

The second mission $M_2$ provides prompt emergency rescue services through warnings and alerts via the coordination of LCSs and ECUs. The scenarios with missions and sub-goals are depicted in Figure 7 and explained in the next section. These are based on disaster management and emergency response operations considering core QAs [68], [72].

#### 1) SCENARIO-A

In the first scenario, the SoS with CPS nodes tries to achieve $M_1$ with sub-goals $G_i$ of monitoring the fire and prediction of real-time events as $G_1M_1$. The IoT nodes comprising humidity, heat, and wind-flow sensors, predict the possible fire event occurrence and generate messages through LCS gateway stations to the ECU. Similarly, for $G_2M_1$, CPS nodes equipped with fire and smoke IoT sensors observe fire events in the area and send information to the ECU through LCS in near real-time without failures. Wind flow sensors also work in conjunction with the detection of fire to provide information about developing fire directions and flows.

#### 2) SCENARIO-B

This scenario is connected to the previous scenario and extends the global mission as $M_2$. In this mission, once the ECU receives information about fire events from CPS sensor nodes and remote nodes (satellite and drones), it starts processing them continuously in real-time and further collaborates with CSs such as police and rescue services, for immediate evacuation and provision of emergency services. It also collaborates with IoT nodes to generate timely signals to contain the fire spread by sending warnings and alerts notifications to fire-fighters.

### D. REQUIREMENTS FOR TRANSIENT AND STEADY-STATES

Based on the scenarios A and B, we have identified core system QAs in relation to performance and reliability, and similarly steady-state requirements for the achievement of SoS missions.

#### 1) PERFORMANCE REQUIREMENTS

Two Performance Requirements (*PR*) for the above scenarios are: 1) latency (events response time), and 2) throughput (number of alerts, warnings sent) per time unit, respectively. Related performance requirements are:

- $PR_1$: What is the probability of data delivery among CPS- nodes, i.e. (sending fire prediction data, receiving sensors data) within time $T$ unit of seconds?
- $PR_2$: The probability that the emergency control unit generates alerts and warnings to first emergency responders within $T$ time unit of seconds after it receives messages from the control station.
- $PR_3$: The probability that smoke and fire data will be delivered to the local control station within the first 10 seconds is greater than 0.90.
- $PR_4$: Fire data will be sent to the ECU and LCS from CPS nodes sensors in less than 70 seconds with a probability greater than 0.80 with more than 50.

#### 2) RELIABILITY REQUIREMENTS

Reliability $R(t)$ of SoS is the likelihood that most of the CSs will be working until the mission has been accomplished at time $t$. SoS missions execute with continuous

time, $t$ predictable failure rates $\Lambda$ i.e. $R(t) = e^{-\Lambda}t$. Core Reliability Requirements (*RR*) are:

- $RR_1$: Likelihood that CSs will be able to complete actions when one of the CSs is degrading (leading to failures) within Time $T$ unit of seconds.
- $RR_2$: There is less than a 50% chance that both CPSFS and CPSHWF will continue to send data successfully.
- $RR_3$: There is more than 50% likelihood that CPSCLS may not send alert messages to the ECU in real-time.

#### 3) STEADY-STATE REACHABILITY

Since the system usually executes on a longer period of time starting from the monitoring of fire to send alerts to the first emergency responders, it is vital to measure steady-states which depict the long-run behavior of the system. We identify few very critical Steady-State Requirements (*SSR*) vital to system success as follows:

- $SSR_1$: The long-run likelihood that various CPS nodes will be able to send the fire situation data from source nodes to nearby nodes?
- $SSR_2$: Steady-state probability that Local control situation will not be able to send data and warnings to nearby nodes and ECU?
- $SSR_3$: What is the long-term possibility of success that mission $M_1$ or mission $M_2$ will be successfully accomplished?

## VII. MODEL-CHECKING SoS USING PRISM

Using CSL, model-checking algorithms based on BTL automatically verify various system states to stochastic behavior and quantitative measures of SoS properties with path and state formulas. However, this creates the problem of state space explosion with the exponential growth of state space [62], [73]. Therefore, it requires a large number of computational resources and time, adding to the complexity. SMC methods try to solve the state-space explosion problem by approximating the sample states using state-space reduction methods [15], [74].

Although there are many SMC tools including UPPAL SPIN and PRISM to verify certain types of probabilistic models [43], [75]. Among these, PRISM offers the features required for the analysis of various types of stochastic models and can optimize the use of resources by state space reduction [43]. Therefore, we use PRISM as our stochastic model verification platform for performing structural and behavioral analysis of SoS. The model specified in HSF is formally transformed into PRISM as the CTMC model, which is further reasoned and analyzed for conducting the architectural analysis. The workflow and core analysis and evaluations to be performed are depicted in Figure 9. SoS Requirements are specified by the system designer for modeling SoS architecture. The stochastic model in HSF transformed through formal rules into the PRISM CTMC model is readily available for various architectural analyses. The SoS CTMC model is analyzed with properties specifications using known

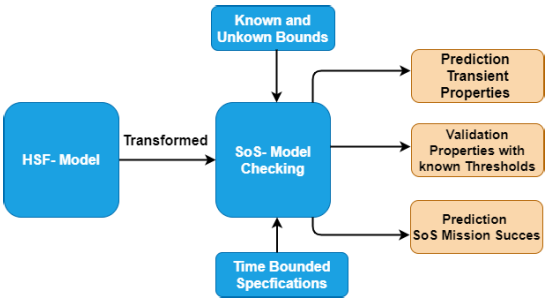



**FIGURE 9.** SoS architecture model-checking from HSF to PRISM.

and unknown bounds to predict mission success and QAs measurements to validate the stated requirements. Results are evaluated considering requirements based on which the SoS architecture model can be refined.

### A. MAPPING RULES

To analyze the system architecture from HSF (syntax and semantics defined in Section V) quantitatively, assess the likelihood of SoS mission success and prediction of QAs in PRISM (defined in Section IIID), an automated transformation approach has been adopted with mathematical principles of mapping. Formally, an architectural model $M$ which is a stochastic model, can be transformed where each member of $M$ is mapped to each modeling element of $\mathcal{S}$: $M \rightarrow \mathcal{S}$. The following mapping rules have been defined for formal translation from HSF to PRISM.

*CS Initial State and Local Variables:* The behavior of SoS starts with an initial state in HSF as $CS_i \in CS$ as a constituent system that corresponds to local state and is represented as Local Variable $LV \in V$ in PRISM and both eventually represent initial state $s_i \in \mathcal{S}$. We form the rule:

$$Rule_1{:}CS_i \in CS \leftrightarrow V \leftrightarrow s_i$$

*CSs and Modules:* Every module and CSs go through a transition with a labeled action with delay rates $r$:

$$Rule_2 : CS_i \in CS \leftrightarrow \mathbb{M} \overset{(\alpha,r)}{\leftrightarrow} s_i \in \mathcal{S}$$

Each CS and module $\mathbb{M}$ also represent state $s_i$ as a result of stochastic action taking place. We can formally represent the CSs' transition w.r.t command in PRISM, which is a result of guard predicates and the transition rate matrix $R$ with input and output (*IO*) ports $(P_{in}, P_{out})$ of interface $I$:

$$Rule_3 : CS_i = R_{i,j}[P_{in}, P_{out}] \leftrightarrow R(s, s')I$$

*Mediators and Modules parallel Composition:* Modules in PRISM can access shared actions $(\alpha_i)$ through parallel composition in the form $(\mathbb{M} \mid (\alpha_i) \mid \mathbb{M}')$, and mediation is performed through the synchronization of commands with global variable $(GV)$ which correspond to mediators responsible for communication among CSs. Interactions are formed on the fly with the ports following the transition as $R$ when $CS_A$ and $CS_B$ form a binding through global variables:

$$Rule_4 : med_i = R_{i,j}[P_{in}\ CS_A \mid CS_B\ P_{out}] \leftrightarrow GV : \mathbb{M} \rightarrow \mathbb{M}' \leftrightarrow R(s, s')$$

*Constraints across coalitions and systems:* Constraints are related to overall coalition behavior and QAs that correspond to commands, which must be valid in terms of respective guards and updates in PRISM modules:

$$Rule_5 : \forall(Col < G_B \wedge Q_A >) = True \leftrightarrow \forall(GV < C > \wedge < \mathcal{R} >)$$

Where *Col* represents a coalition, $G_B$ is a set of global behaviors and $Q_A$ for QAs. Similarly $GV$ with commands ($C$) and rewards ($\mathcal{R}$) are true.

Formally founded rules defined are complete with respect to their compatibility with underlying semantics of HSF and PRISM. A summary of transformation rules is given in Table 1, which contains their descriptions and HSF, PRISM elements aligning one-to-one corresponding through mapping rules. These formal rules enable the automated transformation of the HSF architectural representation to a PRISM CTMC model.

**TABLE 1. Summary of mapping rules for HSF to PRISM formalism.**

| Rules | Description | HSF- Architecture Elements | PRISM Elements |
|---|---|---|---|
| $Rule1$ | System State | CS- State | LV |
| $Rule2$ | Components | CS | Module $\mathbb{M}$ |
| $Rule3$ | States Mapping | $R(s, s')$ $\rightarrow s$(event.r) $\rightarrow s'$ | Command $C$ : $[]guard- > t_r : update;$ |
| $Rule4$ | Parallel Compositions | $Med$= via($p$) & ($p'$) $IO\ Ports$ | $(\mathbb{M} \mid \alpha_i \mid \mathbb{M}')$ |
| $Rule5$ | Coalition Constraints | $\forall(Col < G_B \wedge QA >)$ | $\forall(GV < C > \wedge < R >)$ |

### B. MARKOV MODEL AND VERIFICATION SEMANTICS

The model specified as CTMC requires formal property specification; i.e. a model checker needs to verify that a stochastic model conforms to specified properties according to a certain logic. Before defining the logic specification, we redefine the CTMC with model-checking perspective.

*Definition:* In a model-checking perspective CTMC is defined as a tuple of the form $M =< \mathcal{S}, \mathcal{S}_{init}, \mathbb{Q}, E(s), AP, L(f) >$ where:

- $\mathcal{S}$ and $S_{init\varepsilon\ S}$ are same in Section IIIB.
- $\mathbb{Q}$ is a transition rate matrix, $\mathbb{Q} = [q_{(i,j)}] : \mathcal{S} \times \mathcal{S} \rightarrow \mathbb{R}^+ \geq 0$.
- $E(s)$ is the exit rate, accumulating all rates from going out of state $s$ with action a: $R(s, s') = \sum_{s'} \varepsilon\ S.\ R(s, a, s')$.
- $AP$ and $L(f)$ are the same as defined in Section IIIC.

With model-checking, a CTMC considers current state with state residence time $T_i$ of $X(t)$ random variable and stays $N$ time units in state $i$: $\mathcal{S}[X_i\lambda_i]$ that essentially exhibits a memory-less property and does not require the counting of previous states. For moving from $s_i$ to $s_j$ in time $t$ there exists a transition probability of $1 - e^{R(s_i,s_j).t}$.

This can be expressed as the probability of moving from state $i$ to $j$ with transition rate matrix $R$:

$$R(i, j) = \lambda.\mathcal{P}_{(i,j)} \Rightarrow \mathcal{P}_{(i,j)} = R(s_i, s_j) = \sum_{si \rightarrow s'} R(s_i, s_j)$$

A CTMC model with finite set of states $s_i \in S$ forms a path $\pi$ with time transitions $t \in T = (s_0, a_0, t_o), (s_1, a_1, t_1), \ldots, (s_{n-1}, a_{n-1}, t_{n-1})$ with transitions rate matrix $R(s, s')$:

$$\pi(M) = \pi \in \sum_{i=s} \mathcal{S} \mid \pi = (s, s') \tag{9}$$

Paths and states are used extensively for steady and transient states during the model-checking process overs stochastic state space for system properties verification, which we explain in the section below.

#### 1) CONTINUOUS STOCHASTIC LOGIC

To verify system goals and QAs, the stochastic model of CTMC can be validated quantitatively using CSL descriptions. For a specified period of time or interval, the model's architecture is analyzed in terms of transient properties, while steady-state properties are specified for evaluating long-term system behaviors.

*Definition:* For real-time events $\alpha, AP$ atomic propositions, $\alpha \in AP$, and probability bound is $\mathcal{P} \in [0, 1]$, $t$ is the time interval it takes for moving from $R(s_i, s_{i+1}) \in \mathbb{R}^+ >$ and $\bowtie \in \{\geq, >, \leq, <\}$. The logic to express state $\Phi$ and path $\Psi$ propositions is described as follows:

$$\Phi := \text{true} \mid \alpha \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{P}_{\bowtie P}[\Psi] \mid S_{\bowtie P}[\Psi] \mid \mathcal{R}_{\bowtie x}(\Psi)$$
$$\Psi ::= X \leq t\Phi \mid \Phi U^I \Phi$$

State formulas $\Phi$ are used to verify every state of the system, while path formulas $\Psi$ trace each path. Both of these formulas are based on BTL semantics as described in Section IIIC1. Based on CTL $X$ (Next) and $U$ (until) are temporal operators and can derive other operators such as eventually ($F, \Diamond$) and always ($G, \Box$) for defining advanced logical formulas i.e. $\Diamond^{\leq t}\Phi = true \cup^{\leq t} \Phi$.

#### 2) CSL PATH FORMULAS-SEMANTICS

The path formula $X\Phi^{\leq t}$ is true if $\Phi$ is true in time interval $t$ in path $\pi$ against a state transition. Similarly, $\Phi_1 U^I \Phi_2$, tends to be true if $\Phi_2$ is true in time interval $t$ and $\Phi_1$ also holds on a path execution. Set of paths in $M$ is Path $^M(\pi^M)$ therefore, $s \models \Phi$ assure that state $s$ satisfies property $\Phi$ and Path $\pi^M$: $\pi \models \Psi$ satisfies property $\Psi$. We have transient and steady-state formulas: $s \models \mathcal{P}_{\bowtie p}[\Psi]$ iff $\Psi$ is satisfied in path $\pi$ starting with state $s \in S \in \bowtie_p$ and $\pi \models \mathcal{S}_{\bowtie p}[\Psi]$ iff $\Psi$ is satisfied in long-run for state $s$. The steady-state probability of model $\mathcal{P}_S(M)$ over the the path $\pi^M$ from (6) and (7) with matrix $\mathbb{Q}$ and vector $\mathbb{P}$ are calculated respectively. Therefore, we obtain the initial path $\pi_0(\alpha, s')$ as:

$$\pi(M) = \mathbb{P}\{S = s' | S = (s, t, s')\} = lim_{t\rightarrow\infty}(s, t, s') \tag{10}$$

By applying the system of linear equations, we assume $\pi_i(\alpha, \emptyset) = 0$ to compute steady-state probabilities:

$$\pi(M) = \mathbb{P}(\pi(s).\mathbb{Q} = 0, \sum_{s'} \pi(s, s') = 1 \tag{11}$$

Transient probability is the measure of the probability of a system being in a state $s$ for a particular time interval $t$ $s_i@t$

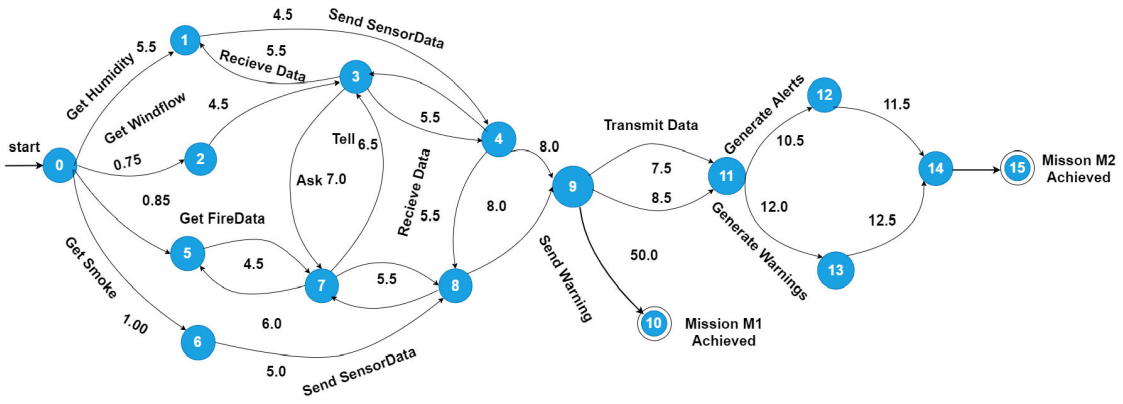



**FIGURE 10.** FM-ERSoS CTMC states transitions and missions with stochastic behaviors in PRISM.

on a path $\pi$ of the form $\pi_{si}@t$; therefore, from(6) and (7) we obtain:

$$\pi(M)_{(s,s',t)} = \mathcal{P}\{\pi_i \in Paths(S)|\pi_{s_i}@t = s'\} \tag{12}$$

System performance and reliability properties are expressed through CSL using (10) and (11) for CTMC models using combinations of CSL operators. Here, we focus on time-bounded verification of properties using 'until' and 'next' operators. The performance and reliability can also be predicted with unknown probability bounds. From (10) and (11) we have general property specification formulas as follows:

$$\mathcal{P}(M \models P_{\bowtie p}(\Psi)) \tag{13}$$

$$\mathcal{P}(M \models S_{\bowtie p}(\Psi)) \tag{14}$$

The quantitative measures for the Markov model of a complex system $M$ can be determined for transient and steady-states overs paths from (13) and (14), respectively, using stochastic logic propositions.

#### 3) BOUNDS: KNOWN AND UNKNOWN

To deal with non-determinism, we adopt a unique approach for the verification of a stochastic model using known and unknown bounds for system properties. CSL allows reachability analysis, also known as Qualitative Reachability, with Known Thresholds (QKT) values to determine if a certain state(s) of the system is reachable with probability ratios. Bound $\bowtie_p$ can be used with relational operators i.e. $\mathcal{P}_{[0,0.7]}(\Psi)$ for $\mathcal{P}_{\leq 0.7}(\Psi)$ to check whether certain thresholds of system properties are met.

Moreover, in CSL, transient and steady-state probabilities can be quantified by leaving the bounds unspecified. Hence, we can define properties in the following way: $\mathcal{P} =?[\Psi] \rightarrow$ (unknown transient probabilities quantification of properties) $\mathcal{S} =?[\Psi] \rightarrow$ (unknown steady-state probabilities quantification of properties). For example, to predict a future state that a stochastic system may reach in a given time interval t with an unspecified probability bound, we apply as: $\mathcal{P} =?[F^{\leq t}]$. The model-checking of system behaviors with unknown bounds is called Quantitative Reachability with Unknown Thresholds (QUT).

*System Qualities with Rewards:* The system model receives a reward for each time instance that it spends in a particular state $s$. For the reward specification, we use the general formula $\mathcal{R}_{\bowtie x}(\Psi)$, $\mathcal{R}$ being the reward amongst a set of paths $\pi$ with starting state $s$ that satisfies $\Psi$ a bound $\bowtie x$. From this the commutative reward $C$ and instantaneous reward $I_r$ are formulated with time unit $t$ as:

$$\mathcal{R}_{\bowtie x}\left(C^{\leq t}\right) \text{ and } \mathcal{R}_{\bowtie x}\left(I_r^{\leq t}\right)$$

## VIII. VALIDATION

The overall process for the verification and validation of the stochastic model is depicted in SAM-SoS in Figures 3 and 9 which show the schematic flow of the approach. The experimental results are checked and verified against stated properties for verification and predication as outputs of the validation process.

### A. STOCHASTIC MODEL IMPLEMENTATION

Based on the transformation rules defined in section VIIA, we have mapped the stochastic SoS CTMC model (FM-ERSoS) from HSF into PRISM semantics. The model consists of abstract CSs forming coalitions at runtime primarily based on Algorithm 1. The stochastic CTMC specification based on the architecture of the SoS model is presented in Figure 10 with possible states and CSs to achieve SoS missions. All the modules (CSs) interact with each other through actions $ACT$ transition rates $T_r$. The complete model specifications of the transformed FM-ERSoS model in PRISM is available in the link.[10]

The model starts from state 0 and moves to multiple states. We assume states {1, 2} and {5,6} represent the CPSHWF and CPSFS nodes respectively. Transitions occur with real-time events and action delay rates. LCS is

[10]https://github.com/ahmadspm/FM-ERSoS-Model-Verification-from-HSF-driven-Models

represented in state 9 as the critical state (there are other possible states for this node, but for simplicity, we have chosen this state). The system reaches to state 10 for completing mission $M_1$ (Generate early warnings and Transmit data to ECU). ECU with states 12 and 13 generates alerts and early warnings to first responders. The absorbing state is 15 and indicates the achievement of mission $M_2$. The parameters used for the model are given in Table 2. $N$ represents system constant, $t_r$ is transition rate, and these rates are extracted by examining the similar type of complex real-time systems [45], [76]. $T$ is the time unit used for a real-time system for model-checking at runtime while individual numbers of CPS nodes are described with minimum and maximum variables.

**TABLE 2. FM-ERSoS stochastic model parameters.**

| Parameter | Description | Possible Values |
|---|---|---|
| ⟨ N⟩ | System Constant | ⟨int>0 ⟩ |
| ⟨ T⟩ | Real-Time Interval units | ⟨seconds, minutes, hours⟩ |
| ⟨ tr1- tr4⟩ | Sensors Reading Rates | ⟨0.5.0.75,0.85,1.00⟩ |
| ⟨ tr5,tr13,tr9⟩ | Sending Sensor Data | ⟨4.5,5.0⟩ |
| ⟨ tr6,tr14,,tr15⟩ | Receiving Sensor Data | ⟨6.5,6.0,8.0⟩ |
| ⟨ tr10,tr11,tr12⟩ | Warnings & Alerts | ⟨8.5,10.5, 12.0⟩ |
| ⟨min, max⟩ nodes | Number of individual CPS nodes | ⟨4,75⟩ |

### B. VERIFICATION THROUGH TEMPORAL LOGIC CSL

By means of CSL, APs are checked for state properties using steady-state and transient analysis via (13) and (14).

#### 1) STEADY-STATE PROPERTIES

Steady-state CSL properties help to verify the long-term behavior of the stochastic FM-ERSoS. By using (14), we formulate steady-state specification as:

$$S = ?[\Phi(Ss \vee Cs)] \tag{15}$$

$\Phi$ is for an AP that could be a single state ($Ss$) or composite state ($Cs$) and $S$ be the steady-state operator.

#### 2) TIME BOUNDED TRANSIENT PROPERTIES AND REWARDS STRUCTURES

To determine how different system QAs perform while achieving certain functionalities in SoS, we use transient CSL properties with time bounds. CSL has two variants of transient time-bounded properties as time bounded next with operator (F) and time bounded until (U). By using (13) we obtain:

$$\mathcal{P} = ?[F^{\leq t}\Phi(Ss \vee Cs)] \tag{16}$$

$$\mathcal{P} = ?[\Phi_1 U^{\leq t}\Phi_2] \tag{17}$$

Here $\Phi$, $\Phi_1$ and $\Phi_2$ are APs for paths states while 't' is the time interval, $q$ is the given threshold value, and the next operator ($F$) may contain single or composite states as APs. Single state is preferred for the 'until' operator ($U$) especially when monitoring a fire from the prediction state to emergency management since CSs are operating in an unstable and volatile environment. Finally, time-based reward structures are used in the model to reason about certain performance and reliability requirements. Following is the general specification for state-based or transition-based rewards verification:

$$\mathcal{R}\{\text{“}Reward''\} = ?[\Phi] \tag{18}$$

Relevant requirements for performing analysis, their description, and logical specifications are presented in Table 3.

**TABLE 3. Transient specifications and related requirements: QUT.**

| Req. | Description | Logical Specification |
|---|---|---|
| $PR_1$ | Fire Data Delivery Rates | $\mathcal{P}$=?[F<=T("Send(A) ∧ Send(B)")] |
| $PR_2$ | Average No. of Alerts & Warnings | $\mathcal{R}${"Alerts-Warnings"}=?[C<=T] |
| $RR_1$ | CPS nodes Reliability | $\mathcal{P}$ =?[¬(op(A)) U <=T(Op(B))] |

### C. RESULTS AND DISCUSSIONS

All the experiments were conducted on Intel(R) i7-7700@ 3.60GHz with a RAM of 16 GB. To perform verification and simulation, a hybrid engine was selected for PRISM using JAVA client as a runtime environment. The maximum iteration for termination was 10000, the probability threshold was established as $1.0 \times 10^{-5}$, and hybrid sparse memory was up to 1024 kB.

#### 1) SYSTEM REQUIREMENTS VERIFICATION THROUGH QKT

We start our experiments by checking the QKT model, which involves qualitative verification of specific system properties. Since the threshold values are already known, results are either satisfied or not satisfied. Table 4 shows the corresponding requirements, CSL logic predicates, and associated outcomes. Most of the requirements are satisfied. However, $PR_4$ does not meet the desired likelihood of 80% which cannot be achieved in real-time due to the uncertain environment in which CPS nodes work. For reliability, especially $RR_2$, results show that there is less chance that both CPS nodes will be able to send data to LCSs.

**TABLE 4. Transient specifications and related requirements: QKT.**

| QKT Req. | CSL logic | Results |
|---|---|---|
| $PR_3$ | $\mathcal{P}$>=0.9[F[0,10] "FireDataSent-LCS"] | Satisfied |
| $PR_4$ | $\mathcal{P}$>=0.8 ["Data from CPS-nodes"U<=70 "FireData-LCS"]>=50 | not Satisfied |
| $RR_2$ | $\mathcal{P}$<=0.5 [F("CPSHWF-willSend") ∧("CPSFS-willSend")] | Satisfied |
| $RR_3$ | $\mathcal{P}$>=0.5[F !"Data from LCS- ECU"] | Satisfied |

#### 2) LONG-RUN SoS BEHAVIOR

Logical specifications for steady-state analysis are derived from (15). Results are presented in Table 5 together with particular requirements and expected values. It is observed that

**TABLE 5.** SoS long-run behavior analysis.

| SSR | Stochastic Logic | Expected Values |
|---|---|---|
| $SSR_1$ | $S$=?["Send_Sensors_Data"] | $7.837e^{-2}$ |
| $SSR_2$ | $S$ =?[¬ ("Local_CS_Transmit_Data")] | $4.459e^{-1}$ |
| $SSR_3$ | $S$=?["(M1_Achieved ∨ M2_Achieved")] | $7.0962e^{-1}$ |

for the initial configuration of CSs (CPSHWF and CPSFS) the probability of sending sensors' data in the long term is quite low at $7.837e^{-2}$. This is because both of these CSs operate in a volatile environment where there is a strong possibility of failure as the fire situation develops. The long-term likelihood that (CPSLCS) will not be able to send messages to the ECU is as high as 45%, indicating that there may be problems with data transmission in future. Once the data has been processed with nodes at the edges i.e. with ECU and a local gateway LCS, the chance of the mission succeeding increases with an expected value of $7.0962e^{-1}$.

### 3) TIME BOUNDED TRANSIENT ANALYSIS WITH QUT

*Fire Data Delivery- Latency: $PR_1$*

One of the important tasks of the fire monitoring system is to measure the real-time fire events from source locations and forward this data to the nearby node in minimum time. The graph in Figure 11 shows the different probabilities of sending sensors data from CPS-nodes to connecting nodes. The time unit was constrained to 20 seconds, while N= 20, 10 and 30. The average number of iterations performed by these nodes was 430. The minimum probability at time unit 1 for CPS nodes CPSHWF and CPSFS is 0.40, and it gradually reaches 0.99. The first two nodes had the maximum amount of response time of five-time units. This indicates that if source CPS nodes are connected well and have the right amount of bandwidth for mediator, the CSs may perform better. When any one of the nodes works with a LCS node, the performance remains moderate as in the second case where the probability that sensor data will be sent from the CPSFS to the gateway station LCS within the given time is more than 60% on average. Similarly, the probability of data delivery from gateway stations to ECU nodes increases to 0.80, indicating the change in likelihood with a positive trajectory. The average response time for the last two pairs of nodes remains at 10-time units, which are quite high. Thus additional nodes are required to reduce the load.

*Alerts and Warnings: $PR_2$*

Once the ECU starts receiving data from local gateways, it is vital that it processes the data immediately to generates the maximum number of warnings and alerts to first emergency responders, including police, ambulance services, and fire-fighters. This can be measured with the reward formula from (18). It obtains the warning data and other events data randomly from distributed LCS and sends messages after further data processing. We verify the combined rewards for alerts and warnings within 30-time units by changing

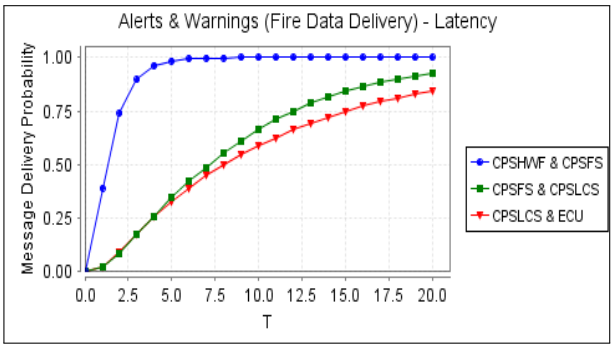



**FIGURE 11.** Latency of CPS nodes in delivering sensors data.

the number of system constants. With $N = 10$, the maximum number of rewards reached is 30, while it reaches 90 with $N = 20$ in given time units, which is the highest reward for sending warnings and alerts messages as depicted in Figure 12. It reflects with increasing N system number of CSs, and it has a direct impact on the performance of ECU as it allows to process more sensors data in and disseminate the information in a timely manner. Depending on the risk areas and geographic spread of the fires, the participating CSs may be increased, starting with source CPS nodes at the local gateway station and moving towards the ECU.

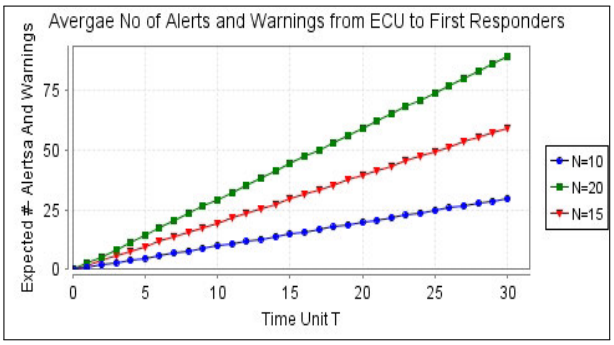



**FIGURE 12.** Average number of alerts and warnings.

*Reliability with CSs Degrading: $RR_2$* Since a SoS comprises various CPS-IoT nodes with multiple independent sensors operating autonomously, the reliability of the system is governed by the number and types of failures occurring among the collaborating CSs operating in a volatile environment. As we can see from the graph in Figure 13 for $RR_2$, when CSs are collaborating, the likelihood that HWF will not be degraded is very low, ranging from (0.039 to 1.52) within time units 1 to 30, respectively. The CPS nodes of the ECU and LCS have a relatively low percentage of failure when collaborating with a maximum value of 0.129 within time $t$. The likelihood that the system will achieve a sub-goal (i.e. ECU can send alerts if LCSs are failing) is relatively lower than desired. During a separate experiment, it was also observed that when predicting the failure of the SoS combined with all CS failures in the long term, i.e. for the next unit of hours,

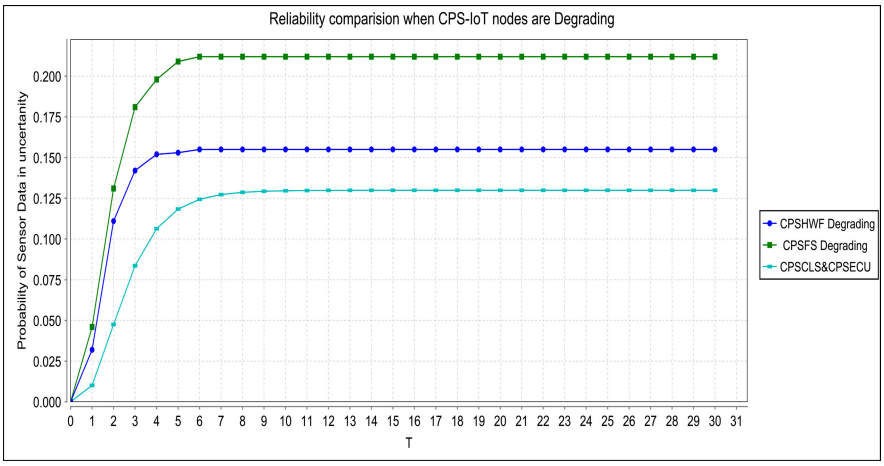



**FIGURE 13.** Reliability of FM-ERSoS with degrading nodes.

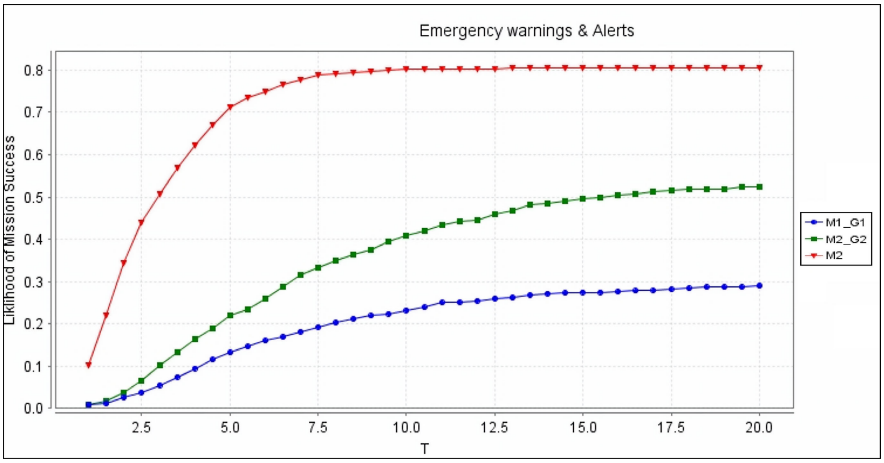



**FIGURE 14.** FM-ERSoS missions predictability to deliver emergency and warning alerts.

the probability was almost 0.99, which is quite high. This is due to unexpected and sudden failures and environmental uncertainty. The reliability of the system can be improved by replacing the degrading CSs with redundant CSs.

#### 4) MISSIONS AND RELATED SUB-GOALS

Moreover, all these analyses enabled us to predict the possible success values for missions and associated goals. Mission $M_1$ has two sub-goals: monitor and predict $G_1M_1$ and transmit data to ECU and send-alerts as $G_2M_1$. Similarly, mission $M_2$ has a collective and time critical mission to generate warnings and send-alerts to first responders. We check the likelihood that these individual missions are achieved within time $T$ units. The property specification logic is provided:

$$\mathcal{P} =?[F <= T\text{“}Mission\text{M-Goal} - GAchieved''] \quad (1)$$

The graph in Figure 14 shows mission behaviors during an emergency and their probability of success in time $T$. There is approximately a 30% chance that mission $M_1$ and its sub-goals will be achieved within the time of 40% success. The probability of achieving mission success $M_1$ is quite high, which reaches to value 0.80 that first responders and population in the area can receive alerts and warnings in a timely manner. However, the level of reliability of participating nodes is determined by the number of failures during the mission.

### D. SUMMARY OF THE RESULTS

The analysis of SoS stochastic architecture behavior in the long term shows that the model has a likelihood of failing as it continues to perform goals. The results reveal that starting with HWF, the long-term stability of CPSFS nodes for fire detection, smoke, and humidity, is expected to be lower than 50%, and there is a greater chance that at certain stages, one or all of the starting CPS nodes may fail once the fire spreads. There are varying results for the quality of attribute performance and reliability of FM-ERSoS. For example, the latency of sensor data exchange from node to node is relatively satisfactory, provided that intermediate nodes are connected. The likelihood that messages will be delivered within

time $T$ units is around 60%, which is satisfactory. This increases as the flow of data go to the next nodes, i.e. from the LCS to ECU and first emergency responders.

The FM-ERSoS reliability is below the expectations, for both the starting and ending CPS nodes. The participating nodes are prone to failures, and it is also observed that there is a strong tendency for individual nodes to degrade. There is a strong probability that CPS nodes CPSFS and CPSHWF will deteriorate with time, with low average reliability of 70%. The LCS gateway may also degrade, further impacting the flow of information while the actual fire situation is escalating. The degradation of the ECU when collaborating with LCS is relatively low as it operates in a much more stable environment. However, collective reliability $R(t)$ decreases as LCS has a higher rate of failures. There is a possibility that more than 55% of the messages fail to reach their destination if one of the CSs is failing within time, $T$. Therefore, all the corresponding nodes must work effectively when collaborating to achieve missions/goals, especially in the case of warnings generated by LCS and sent from the ECU to the first emergency responders. The inevitable failures could lead to disasters; timely evacuations, help services, and rescue may be delayed as the information is lost among nodes.

### 1) IMPROVING PERFORMANCE WITH ADDITIONAL CPS NODES

The performance of FM-ERSoS in terms of response time for mission accomplishment can be improved with additional CSs, especially in emergency cases where human lives and critical infrastructures are under threat [77], [78]. However, additional CSs will increase the cost and may require extra resources for maintainability. We added CSs to the model in parallel based on critical suburban regions with areas of roughly 10-20km. The starting CSs, i.e. CPSFS and CPSHWF, increase in number area-wise when the LCS and ECU are moderate in number.

The graph in Figure 15 shows how additional CSs have influenced the response time in terms of individual fire data delivery. Starting from a suburb with an area of 10 square km, initially, with 10 CSs, it has 200-time units of response time, and it decreases gradually with the addition of other CSs. Similarly, suburban areas of 15 and 20 square km exhibit a similar pattern; however, the impact on response time varies for the specified area. For example, the response time for 15 CSs is 180-time units, which is quite high. In the third case, where the maximum number of CSs is 70, the response time is 55, which is ideal. One important observation is regarding the level of uncertainty: despite the addition of nodes, the overall impact is not always positive in some cases, as shown in the cases of the first and last suburbs with nodes 50 and 30, which have high latencies of 110 and 270 respectively.

### 2) IMPROVING SoS RELIABILITY WITH REDUNDANCY

As discussed above, the reliability of redundant CSs can be improved; however, certain design principles must be followed. The sequential addition of components might not improve the reliability since this will further be lowered if any one of the CSs fails [79]. This problem can be overcome by applying parallel reliability with redundant CSs [80]. Parallel reliability is achieved in terms of unreliability $(1 - R(t))$ of CPS node as:

$$R(t) = 1-[(1-R(t_1))(1-R(t_2))(1-R(t_3))] = 1-\Pi(1-R(t_i))$$

A high collective reliability $R(t)$ value of 0.99 is achieved with the composition of many redundant nodes. However, this approach has associated cost and maintainability overheads for the system in the longer term. Due to the fire spread, initial nodes, i.e. CPSFS and CPSHWF, are prone to failures and can be very volatile. Hence, reliability $R(t)$ is measured with the random addition of parallel CSs. Figure 16 shows the improved reliability with redundancy for the SoS.

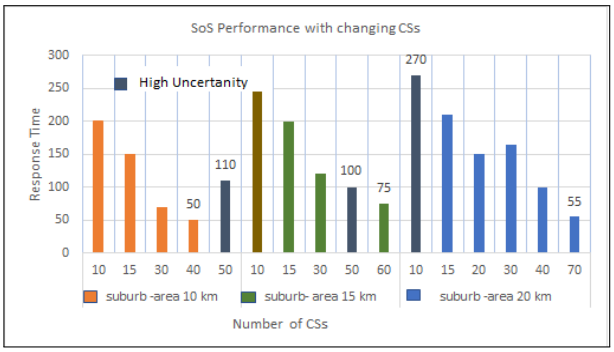



**FIGURE 15.** Improving SoS performance with additional CSs.

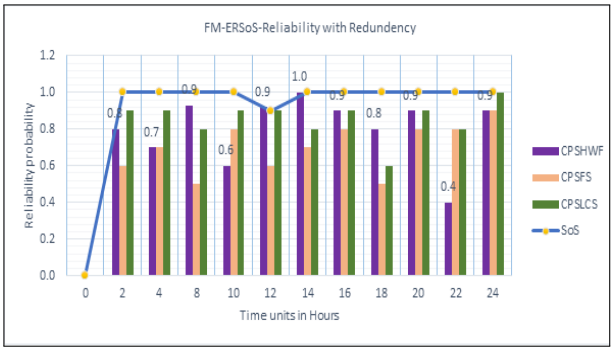



**FIGURE 16.** Improving SoS reliability with redundancy.

### 3) PREDICTION OF MISSIONS SUCCESS

Results reveal that a SoS is prone to mission failures and, if left untested, this would jeopardize the accomplishment of system goals. At the same time, QAs metrics show that reliability and performance need to be improved upon because the values indicate that latency and throughput have a high likelihood of diverging from the stated requirements necessary for mission success. However, with alternative models and rigorous verification, architectural designs can be improved well before time to minimize defects and improve QAs.

**TABLE 6.** Comparative analysis of proposed approach with related work.

| Approach | System Type | Formalism | Model Structure & Behavior | Stochastic Behaviors | Reason Uncertainty | Formal Transformation | Analysis Type | QAs Quantification & Prediction |
|---|---|---|---|---|---|---|---|---|
| Wei et al. [38] | Single System | Automata | +++ | - - | - - | ++ | Transient | safety |
| Huang and Kang [39] | Single System | Semi formal | +++ | Not known | – | - - | Transient | Safety,Security |
| Cavalcante et al. [41] | Single System | CCS | +++ | - - | - - | - - | Transient | Availability |
| Nouri et al. [44] | Single System | Timed Automata | +++ | - - | - - | - - | Transient | Performance |
| Song et al. [47] | Single System | CSP & Z Notation | +++ | ++ | - - | - - | Transient | Performance |
| Mignogna et al. [48] | SoS | Semi formal | +++ | - - | - - | - - | Not known | - - |
| Bozzano et al. [49] | SoS | CSP | +++ | ++ | - - | - - | Not known | Saftey without prediction |
| F.Oquendo [21] | SoS | CCS & CCP | +++ | - - | - - | - - | - - | - - |
| Our Approach. | SoS | Hybrid Formalism | +++ | +++ | +++ | +++ | Steady-state, Transient (known, unknown bounds) | Performance, Reliability |

**Legends**: +++ means: Fully Supported , ++ means: Partially Supported,- - means: Not Supported

## IX. SAM-SoS: COMPARISON WITH EXISTING APPROACHES

Table 6 presents a qualitative comparative analysis of our proposed approach for modeling and verifying SoS architectures with related works. The analysis is performed based on the capabilities of approaches to support the required features: (i) modeling (system type, formalism, structure and behavior, stochastic behaviors, and uncertainty reasoning), and (ii) formal verification (formal transformation, analysis type, and QAs quantified and predicted).

The single-system modeling and verification approaches are able to perform the transient analysis with the quantification of single QA. However, these modeling approaches are unable to deal with the unique architectural characteristics of SoS as these are intended for single stand-alone deterministic systems; therefore, they fail to provide solutions such as that as proposed in our approach to overcome the existing limitations. Among these approaches, QaSten [38] based on Automata by Wei *et al.*, provides a modeling and verification approach with semi-formal transformation rules. Specific to single systems, Song *et al.* [47] have been able to constrain the stochastic behavior of large complex systems to a certain extent.

Among techniques specific to SoS architecture modeling, CML based on CSP in [49] partially supports the description of stochastic behavior using contracts specification coupled with semi-formal SysML notations. Therefore, it is unable to model SoS behaviors that later can be evaluated quantitatively. Sosadl by Flavio [21] is a promising formal modeling language based on CCS and CCP. However, it does not support stochastic models and uncertainty reasoning in its current state since the underlying formalism is unable to constrain actions with probabilities or random rates; and consequently, it fails to provide features necessary for model verification. Although some of these approaches attempt to manage structures and behaviors with probabilistic choice, none of these can deal with the non-determinism and stochasticity of SoS architecture dynamic behaviors for modeling, reasoning, and further quantitative verification. None of these approaches provides formal transformation rules from a stochastic architecture model to a statistical model checker, which should conform semantics to ensure consistency.

Compared to existing techniques, our proposed approach specifically brings modeling capabilities, with HSF deriving process algebraic features that have the essential vocabulary and reasoning semantics to deal with uncertainty and stochastic behaviors of SoS at the architectural level. It allows us to build stochastic models quantitatively, enabling various analyses of SoS architecture models. With formal transformation rules, we provide consistency and completeness for model mapping from HSF to PRISM. Our verification consists of model-checking of steady-state and transient analysis coupled with known and unknown bounds that enable quantitative prediction of multi-QAs in terms of the performance and reliability of SoS architectural models. Our approach is more comprehensive and addresses most of the shortcomings of the current approaches for the modeling and verification of SoS architectures.

## X. CONCLUSION AND FUTURE WORK

In this research paper, we have proposed a comprehensive approach for the modeling and verification of complex SoS architectures. Our contribution to the broader body of knowledge is multi-fold. At the first stage, we have devised a hybrid formalism by integrating syntax and semantics of traditional PAs into SPA. To deal with unpredictable and uncertain

interactions within a SoS, we have introduced stochastic model capabilities. The exogenous behavior of CSs involves parallel actions by means of concurrent stochastic constraints applied to HSF. Improved syntax and semantics give reasoning capabilities for modeling the random, ND behavior of stochastic SoS. In particular, the MDE approach was adopted whereby we transform EBNF into formal DSL, producing an extended HSF for SoS.

At the second stage, for model verification, we establish formal rules for mapping from our stochastic model into the PRISM equivalent transformation, employing SMC. Formal rules allow the automated transformation of the architectural elements of HSF into the PRISM language. Thirdly, we propose a unique verification approach using time bounded modeling checking against labeled Markovian processes. In addition to performing behavioral analysis, we allow system architects to undertake qualitative and quantitative analysis. In this paper, the approach has been validated with an end-to-end case study of real-time CPSoS employed to manage and control emerging fire emergency from a socio-technical perspective. The achievement of the SoS missions and the QAs has been verified through SMC with the application of both steady-state and transient analysis of known and unknown bounds. An adequate number of experiments have been run, and results have been evaluated for the stochastic system at design time before implementation, allowing system architects to make better decisions with alternative choices.

In future work, we intend to improve our formal semantics for HSF in order to build Markov models from CTMCs to MDPs that will improve the non-deterministic reasoning capability of dynamic SoS reconfigurations. For this, we shall add operators to manage dynamic architectural changes at architectural level with certain constraints. By taking into account the complexity of SoS, the MDPs will offer alternative strategies for the assessment of emergent behaviours and QAs at runtime. We plan to increase the ability of QAs by, for example, providing a specification with each dynamic configuration at abstract level. We also plan to perform QAs trade-off analysis of various attributes in order to improve the architectural design decisions for SoS.

## REFERENCES

[1] C. B. Nielsen, P. G. Larsen, J. Fitzgerald, J. Woodcock, and J. Peleska, "Systems of systems engineering: Basic concepts, model-based techniques, and research directions," *ACM Comput. Surv.*, vol. 48, no. 2, p. 41, 2015.

[2] M. W. Maier, "Architecting principles for systems-of-systems," *Syst. Eng.*, vol. 1, no. 4, pp. 267–284, 1998.

[3] H. Kopetz, O. Hoftberger, B. Fromel, F. Brancati, and A. Bondavalli, "Towards an understanding of emergence in systems-of-systems," in *Proc. 10th Syst. Syst. Eng. Conf. (SoSE)*, May 2015, pp. 214–219.

[4] P. Acheson, C. Dagli, and N. Kilicay-Ergin, "Model based systems engineering for system of systems using agent-based modeling," *Procedia Comput. Sci.*, vol. 16, pp. 11–19, Jan. 2013.

[5] A. Tzanev, "Modeling and simulation of systems of systems—A survey," *Cybern. Inf. Technol.*, vol. 13, no. 2, pp. 3–36, 2013.

[6] A. Meilich, "System of systems (SoS) engineering & architecture challenges in a net centric environment," in *Proc. IEEE/SMC Int. Conf. Syst. Syst. Eng.*, Apr. 2006, pp. 1–5.

[7] A. Mohsin and N. K. Janjua, "A review and future directions of SOA-based software architecture modeling approaches for system of systems," *Service Oriented Comput. Appl.*, vol. 12, nos. 3–4, pp. 183–200, Dec. 2018.

[8] N. Jazdi, "Cyber physical systems in the context of industry 4.0," in *Proc. IEEE Int. Conf. Autom., Qual. Test., Robot.*, May 2014, pp. 1–4.

[9] M. Uslar, S. Rohjans, C. Neureiter, F. Pröstl Andrén, J. Velasquez, C. Steinbrink, V. Efthymiou, G. Migliavacca, S. Horsmanheimo, H. Brunner, and T. Strasser, "Applying the smart grid architecture model for designing and validating System-of-Systems in the power and energy domain: A European perspective," *Energies*, vol. 12, no. 2, p. 258, Jan. 2019.

[10] L. Zhang, "Modeling large scale complex cyber physical control systems based on system of systems engineering approach," in *Proc. 20th Int. Conf. Autom. Comput.*, Sep. 2014, pp. 55–60.

[11] M. Jamshidi, "System of systems engineering—New challenges for the 21st century," *IEEE Aerosp. Electron. Syst. Mag.*, vol. 23, no. 5, pp. 4–19, May 2008.

[12] R. N. Taylor, N. Medvidovic, and E. Dashofy, *Software Architecture: Foundations, Theory, and Practice*. Hoboken, NJ, USA: Wiley, 2009.

[13] J. Knodel and M. Naab, "Software architecture evaluation in practice: Retrospective on more than 50 architecture evaluations in industry," in *Proc. IEEE/IFIP Conf. Softw. Archit.*, Apr. 2014, pp. 115–124.

[14] F. Oquendo, "Software architecture challenges and emerging research in software-intensive systems-of-systems," in *Proc. Eur. Conf. Softw. Archit.*, Cham, Switzerland: Springer, 2016, pp. 3–21.

[15] A. Mohsin, N. K. Janjua, S. M. S. Islam, and V. V. Graciano Neto, "Modeling approaches for system-of-systems dynamic architecture: Overview, taxonomy and future prospects," in *Proc. 14th Annu. Conf. Syst. Syst. Eng. (SoSE)*, May 2019, pp. 49–56.

[16] N. Medvidovic and R. N. Taylor, "A classification and comparison framework for software architecture description languages," *IEEE Trans. Softw. Eng.*, vol. 26, no. 1, pp. 70–93, Jan. 2000.

[17] R. J. Allen, "A formal approach to software architecture," Carnegie Mellon Univ., Pittsburgh, PA, USA, Tech. Rep. CMU-CS-97-144, 1997. [Online]. Available: http://reports-archive.adm.cs.cmu.edu/anon/1997/CMU-CS-97-144.pdf

[18] P. Jamshidi, M. Ghafari, A. Ahmad, and C. Pahl, "A framework for classifying and comparing architecture-centric software evolution research," in *Proc. 17th Eur. Conf. Softw. Maintenance Reeng.*, Mar. 2013, pp. 305–314.

[19] B. Marco, C. Jordi, W. Manuel, and B. Luciano, "Model-driven software engineering in practice," *Model-Driven Software Engineering in Practice*, 2nd ed Morgan & Claypool: San Rafael, CA, USA, 2017.

[20] C. A. R. Hoare, *Communicating Sequential Processes*. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 1985.

[21] F. Oquendo, "Formally describing the software architecture of Systems-of-Systems with SosADL," in *Proc. 11th Syst. Syst. Eng. Conf. (SoSE)*, Jun. 2016, pp. 1–6.

[22] J. Fitzgerald, J. Bryans, and R. Payne, "A formal model-based approach to engineering systems-of-systems," in *Collaborative Networks in the Internet of Services*, L. M. Camarinha-Matos, L. Xu, and H. Afsarmanesh, Eds. Berlin, Germany: Springer, 2012, pp. 53–62.

[23] J. Woodcock, A. Cavalcanti, J. Fitzgerald, P. Larsen, A. Miyazawa, and S. Perry, "Features of CML: A formal modelling language for systems of systems," in *Proc. 7th Int. Conf. Syst. Syst. Eng. (SoSE)*, Jul. 2012, pp. 1–6.

[24] P. Lollini, M. Mori, A. Babu, and S. Bouchenak, *AMADEOS SysML Profile for SoS Conceptual Modeling*. Cham, Switzerland: Springer, 2016, pp. 97–127.

[25] A. Clark, S. Gilmore, J. Hillston, and M. Tribastone, "Stochastic process algebras," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Berlin, Germany: Springer, 2007, pp. 132–179.

[26] C. A. Lana, N. M. Souza, M. E. Delamaro, E. Y. Nakagawa, F. Oquendo, and J. C. Maldonado, "Systems-of-systems development: Initiatives, trends, and challenges," in *Proc. XLII Latin Amer. Comput. Conf. (CLEI)*, Oct. 2016, pp. 1–12.

[27] G. Agha and K. Palmskog, "A survey of statistical model checking," *ACM Trans. Model. Comput. Simul.*, vol. 28, no. 1, pp. 6–39, Jan. 2018.

[28] D. C. Schmidt, "Guest Editor's introduction: Model-driven engineering," *Computer*, vol. 39, no. 2, pp. 25–31, Feb. 2006.

[29] L. Bortolussi and A. Policriti, "Hybrid dynamics of stochastic programs," *Theor. Comput. Sci.*, vol. 411, no. 20, pp. 2052–2077, Apr. 2010.

[30] M. Kwiatkowska, G. Norman, and D. Parker, "Stochastic model checking," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Berlin, Germany: Springer, 2007, pp. 220–270.
[31] R. France and B. Rumpe, "Model-driven development of complex software: A research roadmap," in *Future of Software Engineering FOSE*. New York, NY, USA: IEEE Computer Society, May 2007, pp. 37–54.
[32] D. Garlan, "Formal modeling and analysis of software architecture: Components, connectors, and events," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems*. Berlin, Germany: Springer, 2003, pp. 1–24.
[33] Y. Wang, "The real-time process algebra (rtpa)," *Ann. Softw. Eng.*, vol. 14, nos. 1–4, pp. 235–274, 2002.
[34] A. Aldini, M. Bernardo, and F. Corradini, *A Process Algebraic Approach to Software Architecture Design*, 1st ed. London, U.K.: Springer, 2009.
[35] R. Allen, R. Douence, and D. Garlan, "Specifying and analyzing dynamic software architectures," in *Fundamental Approaches to Software Engineering*. Berlin, Germany: Springer, 1998, pp. 21–37.
[36] D. Garlan, R. Monroe, and D. Wile, "Acme: An architecture description interchange language," in *CASCON 1st Decade High Impact Papers*, 2010, pp. 159–173.
[37] P. H. Feiler, D. P. Gluch, and J. J. Hudak, "The architecture analysis & design language (Aadl): An introduction," Carnegie-Mellon Univ Pittsburgh PA Software Engineering Inst, Pittsburgh, PA, USA, Tech. Rep. CMU/SEI-2006-TN-011, 2006.
[38] X. Wei, Y. Dong, and H. Ye, "QaSten: Integrating quantitative verification with safety analysis for AADL model," in *Proc. Int. Symp. Theor. Aspects Softw. Eng.*, Sep. 2015, pp. 103–110.
[39] L. Huang and E.-Y. Kang, "Formal verification of safety & security related timing constraints for a cooperative automotive system," in *Fundamental Approaches to Software Engineering*, R. Hähnle and W. van der Aalst, Eds. Cham, Switzerland: Springer, 2019, pp. 210–227.
[40] J. S. Kim and D. Garlan, "Analyzing architectural styles with alloy," in *Proc. ISSTA Workshop Role Softw. Archit. Test. Anal. ROSATEA*, 2006, pp. 70–80.
[41] E. Cavalcante, J. Quilbeuf, L.-M. Traonouez, F. Oquendo, T. Batista, and A. Legay, "Statistical model checking of dynamic software architectures," in *Software Architecture*, B. Tekinerdogan, U. Zdun, and A. Babar, Eds. Cham, Switzerland: Springer, 2016, pp. 185–200.
[42] A. Bucchiarone and J. P. Galeotti, "Dynamic software architectures verification using dynalloy," *Electron. Commun. EASST*, vol. 10, pp. 1–15, Jun. 2008.
[43] M. Kwiatkowska, G. Norman, and D. Parker, "Prism: Probabilistic symbolic model checker," in *omputer Performance Evaluation: Modelling Techniques and Tools*, T. Field, P. G. Harrison, J. Bradley, and U. Harder, Eds. Berlin, Germany: Springer, 2002, pp. 200–204.
[44] A. Nouri, B. Mediouni, M. Bozga, J. Combaz, S. Bensalem, and A. Legay, "Performance evaluation of stochastic real-time systems with the sbip framework," *Int. J. Crit. Comput.-Based Syst.*, vol. 8, nos. 3–4, pp. 340–370, 2018.
[45] A. Basu, M. Bozga, and J. Sifakis, "Modeling heterogeneous real-time components in BIP," in *Proc. 4th IEEE Int. Conf. Softw. Eng. Formal Methods (SEFM)*, Sep. 2006, pp. 3–12.
[46] B. L. Mediouni, A. Nouri, M. Bozga, M. Dellabani, A. Legay, and S. Bensalem, "Bip 2.0: Statistical model checking stochastic real-time systems," in *Proc. Int. Symp. Automated Technol. Verification Anal.* Cham, Switzerland: Springer, 2018, pp. 536–542.
[47] S. Song, J. Zhang, Y. Liu, M. Auguston, J. Sun, J. S. Dong, and T. Chen, "Formalizing and verifying stochastic system architectures using monterey phoenix," *Softw. Syst. Model.*, vol. 15, no. 2, pp. 453–471, May 2016.
[48] A. Mignogna, L. Mangeruca, B. Boyer, A. Legay, and A. Arnold, "Sos contract verification using statistical model checking," in *Proc. 1st Workshop Adv. Syst. Syst., AiSoS*, vol. 133, K. G. Larsen, A. Legay, and U. Nyman, Eds. Rome, Italy, Mar. 2013, pp. 67–83.
[49] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, "The compass approach: Correctness, modelling and performability of aerospace systems," in *Proc. Int. Conf. Comput. Saf., Rel., Secur.* Berlin, Germany: Springer, 2009, pp. 173–186.
[50] Y. Y. Haimes, "Modeling complex systems of systems with phantom system models," *Syst. Eng.*, vol. 15, no. 3, pp. 333–346, Sep. 2012.
[51] W. C. Baldwin and B. Sauser, "Modeling the characteristics of system of systems," in *Proc. IEEE Int. Conf. Syst. Syst. Eng. (SoSE)*, May 2009, pp. 1–6.
[52] D. Richards, B. D. McKay, and W. A. Richards, "Collective choice and mutual knowledge structures," *Adv. Complex Syst.*, vol. 1, nos. 2–3, pp. 221–236, Jun. 1998.
[53] J. Boardman and B. Sauser, "System of systems–the meaning of of," in *Proc. IEEE/SMC Int. Conf. Syst. Syst. Eng.*, Apr. 2006, p. 6.
[54] A. W. Wymore, *Model-Based Systems Engineering*. Boca Raton, FL, USA: CRC Press, 2018.
[55] Y. Y. Haimes, *Modeling and Managing Interdependent Complex Systems of Systems*. Hoboken, NJ, USA: Wiley, 2018.
[56] D. Shin, "A socio-technical framework for Internet-of-Things design: A human-centered design for the Internet of Things," *Telematics Informat.*, vol. 31, no. 4, pp. 519–531, Nov. 2014.
[57] M. J. C. de Henshaw, "Systems of systems, cyber-physical systems, the Internet-of-Things... whatever next," *Insight*, vol. 19, no. 3, pp. 51–54, 2016.
[58] A. Sokolova and E. P. De Vink, "Probabilistic automata: System types, parallel composition and comparison," in *Validation of Stochastic Systems*. Berlin, Germany: Springer, 2004, pp. 1–43.
[59] Z. Manna and A. Pnueli, *The Temporal Logic of Reactive and Concurrent Systems: Specification*. New York, NY, USA: Springer, 2012.
[60] R. Alur, T. A. Henzinger, and O. Kupferman, "Alternating-time temporal logic," *J. ACM (JACM)*, vol. 49, no. 5, pp. 672–713, Sep. 2002.
[61] E. A. Emerson and J. Srinivasan, "Branching time temporal logic," in *Linear Time, Branching Time Partial Order Logics Models for Concurrency*, J. W. de Bakker, W. P. de Roever, and G. Rozenberg, Eds. Berlin, Germany: Springer, 1989, pp. 123–172.
[62] M. Ahmad, N. Belloir, and J.-M. Bruel, "Modeling and verification of functional and non-functional requirements of ambient self-adaptive systems," *J. Syst. Softw.*, vol. 107, pp. 50–70, Sep. 2015.
[63] R. Alur and T. A. Henzinger, "Reactive modules," *Formal Methods Syst. Des.*, vol. 15, no. 1, pp. 7–48, Jul. 1999.
[64] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with PRISM: A hybrid approach," *Int. J. Softw. Tools Technol. Transf.*, vol. 6, no. 2, pp. 128–142, Aug. 2004.
[65] V. A. Saraswat and M. Rinard, "Concurrent constraint programming," in *Proc. 17th ACM SIGPLAN-SIGACT Symp. Princ. Program. Lang.*, 1989, pp. 232–245.
[66] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Model-checking continuous-time Markov chains," *ACM Trans. Comput. Log.*, vol. 1, no. 1, pp. 162–170, Jul. 2000.
[67] Á. Asensio, T. Blanco, R. Blasco, Á. Marco, and R. Casas, "Managing emergency situations in the smart city: The smart signal," *Sensors*, vol. 15, no. 6, pp. 14370–14396, Jun. 2015.
[68] N. Suri, Z. Zielinski, M. Tortonesi, C. Fuchs, M. Pradhan, K. Wrona, J. Furtak, D. B. Vasilache, M. Street, V. Pellegrini, G. Benincasa, A. Morelli, C. Stefanelli, E. Casini, and M. Dyk, "Exploiting smart city IoT for disaster recovery operations," in *Proc. IEEE 4th World Forum Internet Things (WF-IoT)*, Feb. 2018, pp. 458–463.
[69] A. Bondavalli, S. Bouchenak, and H. Kopetz, *Cyber-Physical Syst. Systems: Foundations—A Conceptual Model Some Derivations: AMADEOS Legacy*. Cham, Switzerland: Springer, 2016, vol. 10099.
[70] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, "Design, modelling, simulation and integration of cyber physical systems: Methods and applications," *Comput. Ind.*, vol. 82, pp. 273–289, Oct. 2016.
[71] S. K. Khaitan and J. D. McCalley, "Design techniques and applications of cyberphysical systems: A survey," *IEEE Syst. J.*, vol. 9, no. 2, pp. 350–365, Jun. 2015.
[72] A. M. Al-Smadi, M. K. Alsmadi, A. Karim Baareh, I. Almarashdeh, H. Abouelmagd, and O. S. Shidwan Ahmed, "Emergent situations for smart cities: A survey," *Int. J. Electr. Comput. Eng. (IJECE)*, vol. 9, no. 6, p. 4777, Dec. 2019.
[73] A. Bianco and L. de Alfaro, "Model checking of probabilistic and nondeterministic systems," in *Foundations of Software Technology and Theoretical Computer Science*, P. S. Thiagarajan, Ed. Berlin, Germany: Springer, 1995, pp. 499–513.
[74] M. Kwiatkowska, G. Norman, and D. Parker, "Probabilistic symbolic model checking with prism: A hybrid approach," in *Tools and Algorithms for the Construction and Analysis of Systems*, J.-P. Katoen and P. Stevens, Eds. Berlin, Germany: Springer, 2002, pp. 52–66.

[75] J.-P. Katoen, "Advances in probabilistic model checking," in *Verification, Model Checking, Abstract Interpretation*, G. Barthe and M. Hermenegildo, Eds. Berlin, Germany: Springer, 2010, p. 25.
[76] J. Nilsson, B. Bernhardsson, and B. Wittenmark, "Stochastic analysis and control of real-time systems with random time delays," *Automatica*, vol. 34, no. 1, pp. 57–64, Jan. 1998.
[77] M. F. N. Maghfiroh, M. Hossain, and S. Hanaoka, "Minimising emergency response time of ambulances through pre-positioning in dhaka city, bangladesh," *Int. J. Logistics Res. Appl.*, vol. 21, no. 1, pp. 53–71, Jan. 2018.
[78] S. Djahel, N. Smith, S. Wang, and J. Murphy, "Reducing emergency services response time in smart cities: An advanced adaptive and fuzzy approach," in *Proc. IEEE 1st Int. Smart Cities Conf. (ISC)*, Oct. 2015, pp. 1–8.
[79] L.-X. Xu, C.-F. Xie, and L. Xu, "Reliability analysis on parallel system with n element obeying exponential distribution," in *Genetic and Evolutionary Computing*. Cham, Switzerland: Springer, 2015, pp. 409–418.
[80] D. W. Coit, "Maximization of system reliability with a choice of redundancy strategies," *IIE Trans.*, vol. 35, no. 6, pp. 535–543, Jun. 2003.

**AHMAD MOHSIN** received the B.S.C.S. degree from BZU, Multan, and the M.S. degree from FAST-NUCES, Islamabad, Pakistan. He is currently pursuing the Ph.D. degree with the School of Science, Edith Cowan University (ECU), Australia. He has contributed towards the Systems Architecture Enhancement for Real-Time Systems with local Australian Software Industry in a research project. His current research interests include software architecture, modeling and simulation of complex systems, cyber-physical systems, and the IoT. He was a recipient of the ECU Higher Degree from the Research Scholarship.

**NAEEM KHALID JANJUA** (Member, IEEE) was a Postdoctoral Fellow with UNSW Canberra and a Research Associate with Curtin University. He is currently a Senior Lecturer with the School of Science, Edith Cowan University, Perth. He works in business intelligence and web-based intelligent decision support systems. He has published an authored book, a book chapter, and various articles in international journals and refereed conference proceedings. His research interests include defeasible reasoning, argumentation, ontologies, data modeling, cloud computing and the IoT, machine learning, and data mining. He serves as an Associate Editor for the *International Journal of Computer System Science and Engineering (IJCSSE)* and the *International Journal of Intelligent Systems (IJEIS)*.

**SYED M. S. ISLAM** (Member, IEEE) received the B.Sc. degree in electrical and electronic engineering from the Islamic Institute of Technology, in 2000, the M.Sc. degree in computer engineering from the King Fahd University of Petroleum and Minerals, in 2005, and the Ph.D. degree (Hons.) in computer engineering from The University of Western Australia (UWA), in 2011. He was a Research Assistant Professor with UWA from 2011 to 2015, a Research Fellow with Curtin University from 2013 to 2015, a Casual Lecturer with UWA and Curtin University from 2015 to 2016, and a Lecturer in 2016. He was also an Adjunct Lecturer with UWA from 2016 to 2019. He has published over 60 research articles. He has supervised three H.D.R. students and supervising 12 others. His research interests include artificial intelligence, computer vision, pattern recognition, big-data analysis, bio-metrics, medical imaging, the Internet of Things (IoT), image processing, and biomedical engineering. He was a Technical Committee Member of 25 conferences. He serves seven professional bodies, including the IEEE and the Australian Computer Society (Senior Member). He received 15 public media releases, including a TV News Story and four live radio interviews. He also received the NHMRC Ideas Grant 2019 AUD (467K) and nine other external research grants. He serves as an Associate Editor for IEEE Access and a Guest Editor for *Healthcare*. He served as a Regular Reviewer for 26 journals.

**MUHAMMAD ALI BABAR** was a Reader of software engineering with Lancaster University. He is currently a Professor with the School of Computer Science, The University of Adelaide. He is an Honorary Visiting Professor with the Software Institute, Nanjing University, China. He is also the Director of Cyber Security Adelaide (CSA), which incorporates the Cyber Security Cooperative Research Centre (CSCRC), whose estimated budget is around 140 million over seven years with 50 million provided by the Australia Government. In Software Engineering Education, he led the University's effort to redevelop the Bachelor of Engineering (software) degree that has been accredited by the Australian Computer Society and the Engineers Australia (ACS/EA). He spent almost seven years in Europe (Ireland, Denmark, and U.K.) as a Senior Researcher and an Academic. He has established the Interdisciplinary Research Centre and the Centre for Research on Engineering Software Technologies (CREST), where he leads the research and research training of more than 20 (12 Ph.D. students) members. Apart from his work, he has industrial relevance as evidenced by several research and development projects and setting up a number of collaborations in Australia and Europe with industry and government agencies. His publications have been highly-cited within the discipline of software engineering as evidenced by his H-index is 46 with 8240 citations as per Google Scholar in January 2020. He leads the theme on Platform and Architecture for Cyber Security as a Service with the Cyber Security Cooperative Research Centre. He has authored/coauthored more than 220 peer-reviewed publications through premier Software Technology journals and conferences.