Research Article

Ilyas Fakhir*, Asad Raza Kazmi, Awais Qasim, and Atif Ishaq

# SMACS: A framework for formal verification of complex adaptive systems

https://doi.org/10.1515/comp-2022-0275
received May 6, 2021; accepted April 14, 2023

**Abstract:** Self-adaptive systems (SASs) have the capability to evaluate and change their behavior according to changes occurring in the environment. Research in this field is being held since mid-60, and over the last decade, the importance of self-adaptivity is being increased. In the proposed research, colored petri nets (CPN) formal language is being used to model self-adaptive multiagent system. CPN is increasingly used to model self-adaptive complex concurrent systems due to its flexible formal specification and formal verification behavior. CPN being visually more expressive than simple, Petri Nets enable diverse modeling approaches and provides a richer framework for such a complex formalism. The main goal of this research is to apply self-adaptive multi-agent concurrent system (SMACS) for complex architectures. In our previous research, the SMACS framework is proposed and verified through traffic monitoring system. All agents of SMACS are also known as intelligent agents due to their self-adaptation behavior. Due to decentralized approach in this framework, each agent will intelligently adapt its behavior in the environment and send updates to other agents. In this research, we are choosing smart computer lab (SCL) as a case study. For internal structure of each agent modal, $\mu$-calculus will be used, and then a model checker TAPAs: a tool for the analysis of process algebras will be applied to verify these properties. CPN-based state space analysis will also be done to verify the behavioral properties of the model. The general objective of the proposed system is to maximize the utility generated over some predetermined time horizon.

*** Corresponding author: Ilyas Fakhir,** Department of Computer Science, GC University, Lahore, Pakistan, e-mail: fakhir@gcu.edu.pk
**Asad Raza Kazmi:** Department of Computer Science, GC University, Lahore, Pakistan, e-mail: arkazmi@gcu.edu.pk
**Awais Qasim:** Department of Computer Science, GC University, Lahore, Pakistan, e-mail: awais@gcu.edu.pk
**Atif Ishaq:** Department of Computer Science, GC University, Lahore, Pakistan, e-mail: atif.ishaq@gcu.edu.pk

**Keywords:** formal specifications, formal verification, intelligent agents, complex system

## 1 Introduction

The system adaptivity has been studied extensively since mid-60s, and new approaches to explicate the elementary principles of self-adaptivity are being found with significant efforts of scientific community. Over the last decade, the literature about self-adaptivity has been extensively used, and different types of concepts and interpretations are being proposed by different researchers. In any evolving scientific research field, there is a chance of instability, the conceptual evolution of the system under investigation. That is, the instability may get induced when the model under study evolves conceptually. The investigation links with the development of the model at different stages rather than the treatment of the subject as whole. However, every scientific research may play an important role to support uniform global understanding theory.

The importance of self-adaptivity in various related areas is being progressively more acknowledged over the past decades. Software engineering and other related fields like requirements engineering [1,2], middleware [3,4], software architecture [5–7], and component-based architecture development [8,9] are very common features to introduce self-adaptivity successfully. For the development of self-adaptive systems (SASs), software engineering approaches would perfectly be applicable across multiple domains [10]. The consistency of the system must be assured during the adaptation process [11]. Assurance is required for both functional and nonfunctional properties, i.e., [12]. Guaranteeing these properties at runtime in SASs is particularly challenging due to the varying assurance needs posed by a changing system or execution environment, both fraught with uncertainty [13,14]. Nevertheless, the properties specified in the system requirements need to hold before, during, and after adaptation [11,15]. Cheng et al. proposed a model to address assurance of SAS with name M@RT [16]. Recent approaches

recognize the need to produce, manage, and maintain software models all along the software's life time to assist the realization and validation of system adaptations while the system executes [17–19].

A SAS is an improved version of software-intensive system having the ability to quickly respond to changes occurring in their environment or autonomously system can adapt its structure and behavior at run-time [20]. This idea presents shifting the human role from operational to strategic because high level policies of human state how a system should react if changes occur and then system adapt these changes at run-time autonomously [21]. Similarly, in 2011, Weiss et al. proposed a self-adaptive embedded system having ability to make decisions at runtime on adaptivity in [22].

From the area of control theory, Müller et al. proposed feedback control loops in [23], which are very important features in engineering self-adaptive software systems. Such control loops are being organized by four components that are responsible for the fundamental functions of self-adaptation: Monitor, Analyze, Plan, and Execute (MAPE), often stated as the MAPE loop as in the IBM architecture blueprint. These functions are referred to as collect, analyze, decide, and act by Dobson et al. in [24].

Colored Petri Nets (CPNs) [25] are powerful mathematical modeling language to model complex systems. A brief introduction to the complex system is presented in ref. [26]. Due to graphical notation and very powerful analysis methodology, CPN play, an intensive role to model SASs as well as concurrent systems and biological systems [27,28]. Through CPN tool, small modules can be constructed for the specification of large one. By the help of well-defined set of interfaces, all these modules interact with each other. Context adaptation is a new perspective in the field of self-adaptation. Context-awareness is a significant part of today's research of ubiquitous computing applications. The behavior of such applications is generally described by implanting the clarification of contextual information inside applications and having facility to reuse this information by other applications. Such type of applications also behaves like multi-agent system (MAS), where each context can be represented as an agent having its own behavior as discussed in ref. [29]. To model context-aware systems by using ordinary Petri Net (PN) and showing more expressiveness by other extended PN classes is proposed in ref. [30]. These agents update their behavior by their internal contextual information and update their neighboring agents, too. PN is relatively more suitable to model such type of complex systems. Due to the uncertainty of computational environments and complex software systems, new motivations occur to explore new modeling and managing techniques of systems and services in the fields such as control theory, artificial intelligence, and biology. In this regards, the self-adaptivity is the most intensive research direction. Macías-Escrivá et al. provides some well-known definitions in their recent survey in [31].

Two types of approaches are used for modeling SASs; top-down approach; and bottom-up approach. In the top-down SASs the control is centralized and behave with guidance of central controller, which evaluates its surroundings behavior and adapts itself accordingly. While, bottom-up SASs having decentralized phenomena, such type of systems having self-adaptive cooperation or self-organization. Bottom-up approach behaves like MAS, because in MAS, each agent updates its own behavior and also updates the environment that guides other agents. The key features of MAS in the engineering of SASs are, specifically, loose coupling, context sensitivity, robustness in response to failure, and unexpected concurrent events. Weyns and Georgeff proposed goal-oriented loose coupling of agents in [32], which provides the flexibility for modeling self-adaptivity and reuse.

In this research work, we are trying to check our self-adaptive multi-agent concurrent system (SMACS) framework proposed in [33] for more complex system than previous one. For implementing SMACS framework, we chose smart computer lab (SCL) as a case study. In proposed work, we are trying to improve an idea of MAPE-K-based self-adaptive framework in a more expressive way by the help of formal specification and verification [34]. PNs are being used for modeling concurrent systems for the last few decades, but according to our knowledge, PNs are not commonly used for self-adaptive-based concurrent systems. So we have been using CPN for modeling self-adaptive MAS, because CPN is more expressive than simple PNs to model such complex systems having concurrent behavior. For example, the processes are represented as tokens in PNs, and in complex systems, processes are distinguishable with respect to their properties and behaviors. So, it only possible to model such systems with CPN, because we can differentiate the processes to assign different colors to them, and it is not possible in simple PN. All agents of SMACS are also known as intelligent agents due to their self-adaptive behavior. So, to verify self-adaptive behavior of each agent, we applied modal $\mu$-calculus ($\mathcal{M}_\mu$).

In the complex system, components are physically and functionally interconnected in heterogeneous way. This heterogeneity of components is based on structural or dynamic complexity as proposed in ref. [35], by which highly interconnected dependent and interdependent

components integrate with each other. Moreover, the dynamic complexity leads to abrupt change in the system's behavior due to the change in the operational and environmental conditions of the interconnected systems. The main idea behind this work is to achieve true concurrency of multiple interconnected self-adaptive agents with their dynamic behavior. Here, true concurrency means multiple atomic computing tasks can take place at one step, on contrary an interleaving-based concurrent behavior enforce that only a single task in a parallel composition can execute an atomic action. A more appropriate mathematical rule for measuring true concurrency is represented in equation (1). By the help of true concurrency fairness, property can be measured easily. PNs is one of the best formal approaches to express true concurrent systems. Formal modeling approach is more expressive because the traditional approaches like activity diagrams and transition systems may be the candidates to express our system, but these methods fail to generate counter example due to their nonexhaustiveness. While, on the other hand, the formal modeling approaches are more suitable to model the dynamical systems like SMACS. Using full expressive power of a logic often leads to more direct and concise statements of properties but requires more human interaction to guide the proofs. So, formal modeling approach is more expressive for proving liveness and safety properties of true concurrent systems, as well as to verify correctness and consistency of concurrent systems. A CPN-based model having concurrent behavior is proposed in [36], in which safety and bounded properties are verified through intuitionistic linear time $\mu$ calculus (I$\mu$TL) [37].

$$\frac{P_1 \to P_1', P_2 \to P_2', \ldots, P_n \to P_n'}{P_1|P_2|\ldots|P_n \to P_1'|P_2'|\ldots|P_n'}. \tag{1}$$

This equation shows that a process interface executes multiple processes from $P_1$ to $P_n$ in parallel, where the fractional part is represented as $\frac{\text{Input port}}{\text{Output port}}$ and the parallel composition of all processes to execute an atomic action is represented as $P_1|P_2|\ldots|P_n$.

The remaining article is organized as follows: in Section 2, preliminaries for proposed modeling techniques are discussed; in Section 3, a case study for implementing proposed model is discussed; in Section 4, analysis and results are discussed; and finally, conclusion and future work are discussed in Section 5.

# 2 Preliminaries

SASs having dynamic concurrent behavior presented in ref. [38], by which systems adapt behavior dynamically. So, PNs are relatively better formal approach for modeling systems, which simulate dynamically as in ref. [39]. PNs combine a well-defined mathematical theory with a graphical representation of the dynamic behavior of systems. The theoretic aspect of PNs allows precise modeling and analysis of system behavior, while the graphical representation of PNs enable visualization of the modeled system state changes. This combination is the main reason for the use of PNs. While the static logic based approaches including prepositional and predicate logics do not express dynamical systems. The dynamical and temporal logics-based methods like linear temporal logic, computation tree logic (CTL) or $\mu$-calculus could model dynamical systems by using their mathematical techniques. These methods lack in expressing the system behavior figuratively, while PNs have both capabilities of expression. CPN having flexibility to model complex dynamic system and to attaining true concurrency during execution of such complex model is also possible by CPN, which having all basic properties of simple PN including its color sets and guard functions. In this section, we will discuss terminologies use to help for formal modeling, specification, and analysis of self-adaptive MAS.

## 2.1 Multi-agent system

MAS is an important field of software engineering that comprises two main concepts of agents' interaction. All agents interact with each other either through top-down approach or bottom-up approach. In top-down approach, all agents interact with each other by a centralized mechanism indirectly, on contrary through bottom-up approach, the interaction between agents is decentralized, and each agent is directly interacted to each other agent during any change in their environment. An agent can be composed of reactive, deliberative, or a hybrid architecture of both. In the indirect approach, an agent can modify the environment of other agents through initialization of an action. In this approach, some agents can share a subpart of the environment. In MAS, every agent occupies their own environment, and somehow, it is related to the Meta-level environment as shown in Figure 1. This phenomenon is also known as open environment, and a complex situation is created in this environment due to dynamic behavior of unknown components in future. This structure presented in [40] changes over time and becomes heterogeneous in nature. Qasim et al. provide an idea of formal specification and verification of real-time MAS using Timed-Arc PN in [41].

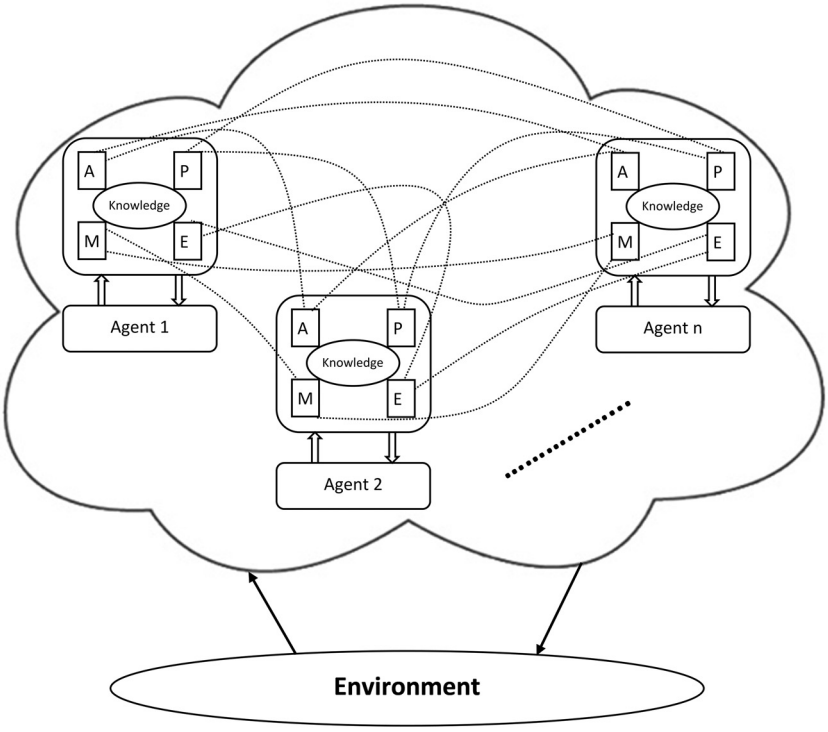


**Figure 1:** Interaction between agents in MAPE-K based meta-level environment.

An intelligent agent by Wooldridge in [42] is another important basic element of MAS, having classical characteristics such as reactivity, pro-activeness, and social ability. In reactivity, agents have ability to response timely according to design objectives if any change occurs in their environment. In pro-activeness, agents demonstrate their goal-oriented behavior to satisfy design objectives. Negotiation between agents is their social ability according to design objectives. So, these characteristics represent the significance of MAS to develop a system for complex architecture. In the engineering of SASs, MASs have some key features such as context-sensitivity, loose coupling, unexpectedness, and robustness in response to failure.

## 2.2 Self-adaptive system

SASs have ability to adapt its behavior according to change in its environment as discussed in [43]. It is the behavior of SAS to organize itself autonomously by accommodating changes in its environment and context. Some SASs may have ability to adapt changes in environment without human interaction through some higher-level policies as presented in [10]. Just like agents in MAS, adaptation is a process for switching between models in the multi-model system. So it is the capability of a system to adapt the behavior of the environment by selectively switching and executing between models. Formal verification of MAPE-K-based real-time MAS is proposed in ref. [44]. In order to improve the performance of a complex system, self-adaptivity plays a key role as the system learns to change behavior on its own. High-level dependability, adaptivity, robustness, and availability for developing self-adaptive software are proposed in ref. [45].

The adaptation mechanism is decomposed by Selehie and Tahvildari in 2011 into several processes: monitoring the environment and software objects (i.e., context-awareness and self-awareness), analyzing substantial variations, planning of reaction, and executing the decisions.

For the specification of SASs, monitoring, and switching between adaptive behaviors, goal-oriented models have proven their effectiveness. The ability to reason about partial goal satisfaction is a strength of goal-oriented modeling. Most of the existing self-adaptive models are taking benefits of the Sensor-Plan-Act model, extensively used in modeling traditional robotic systems presented in ref. [46]. In such systems, events are collected, analyzed, and attached to update the global model, and then in updated situation, each entity plans its strategy.

## 2.3 Colored PNs

In CPNs [25], tokens are in the form of colors denoting specific data types. Any transition is called enabled for

firing when all of its input places having valid colored tokens, and valid arc bindings are used to produce the necessary colored tokens to its output places. The firing of transitions in CPN can modify the data values of these coloured tokens and change the behavior of model.

The formal definition of CPN is as follows: CPN = $(\Sigma, P, T, A, V, C, G, E, I)$, where:

(1) $\Sigma$ is a finite set of nonempty color sets.
(2) $P$ is a finite set of places.
(3) $T$ is a finite set of transitions.
(4) $A$ is a set of directed arcs such that: $A \subseteq P \times T \cup T \times P$.
(5) $V$ is a typed finite variables set such that: Type$[v] \in \Sigma, \forall v \in V$.
(6) $C$ is a color function (nonempty color set). It is defined as follows: $C : P \to \Sigma$.
(7) $G$ is a guard function defined as: $G : T \to \text{EXPR}_v$. It assigns a guard to each transition $t$ such that: $\forall t \in T$: [Type$(G(t))$ = Bool].
(8) $E$ is an arc expression function, defined as $E : A \to \text{EXPR}_v$ such that $\forall a \in A$ : [Type$(E(a))$ = $C(p)_{MS}$], where $p$ is the place connected to arc $a$.
(9) $I$ is an initialization function, defined as $I : P \to \text{EXPR}_\phi$ such that: $\forall p \in P$ : [$Type(I(p))$ = $C(p)_{MS}$].

CPN Meta language (ML) is a Standard ML (SML)-based functional programming language. CPN ML having all characteristics SML language with extension of defining color set and declaring variables. SML having more expressiveness to model interconnected systems in complex environment. It is also a best approach for state space analysis and performance analysis. In the present research, we will first applied SMACS framework to more complex real-time self-adaptive concurrent model and then verify behavioral properties through the state-space analysis.

## 2.4 Modal $\mu$-calculus ($\mathcal{M}_\mu$)

Modal $\mu$-calculus is the extension of Hennessy-Milner Logic [47] with least fixed-point operator "$\mu$" and greatest fixed-point operator "$\nu$," and it is more suitable to achieve true concurrency. Modal $\mu$-calculus is also a fundamental temporal logic applied to model the context of temporal properties of systems and to model infinite behavior of concurrent systems. Many expressive temporal logics such as propositional dynamic logic, CTL, and CTL* are also the fragments of Modal $\mu$-calculus [48]. Intuitively, $\mu$-calculus has the capability to express the modalities into recursive patterns. The syntax of modal $\mu$-calculus is given in equation (2):

$$\begin{aligned}\mathcal{F}_\mu \ ::= \ & \mathcal{Z}_\mu | \top_\mu | \ \bot_\mu \mid \neg \mathcal{F}_\mu | \mathcal{F}_{\mu 1} \wedge \mathcal{F}_{\mu 2} | \mathcal{F}_{\mu 1} \vee \mathcal{F}_{\mu 2} | \mathcal{F}_{\mu 1} \\ & \to \mathcal{F}_{\mu 2} | \langle \beta \rangle \mathcal{F}_\mu | [\beta] \mathcal{F}_\mu | \mu \mathcal{Z}_\mu . \ \mathcal{F}_\mu | \nu \mathcal{Z}_\mu . \ \mathcal{F}_\mu .\end{aligned} \tag{2}$$

The aforementioned syntax of $\mathcal{M}_\mu$ represents that $\mathcal{F}_\mu$ is a formula that satisfies different modalities, where $\mathcal{Z}_\mu$ is a set of states also called propositional variables and $\beta \subseteq Act$, also known as $\beta$-transition. $\top_\mu$ is the formula that holds universally, $\bot_\mu$ holds nowhere, and negation, conjunction, disjunction, and implication have their usual meaning. Modalities are expressed by transitions, as formula $\langle \beta \rangle \mathcal{F}_\mu$ holds if there is an outgoing $\beta$-transition to some states satisfy $\mathcal{F}_\mu$, similarly formula $[\beta]\mathcal{F}_\mu$ holds if there are all outgoing $\beta$-transition states satisfy $\mathcal{F}_\mu$. $\mu \mathcal{Z}_\mu . \mathcal{F}_\mu$ and $\nu \mathcal{Z}_\mu . \mathcal{F}_\mu$ represents the least fixed point and greatest fixed-point for $\mathcal{Z}_\mu$ set of states by applying $\mathcal{F}_\mu$ formulas.

### 2.4.1 Semantics of $\mathcal{M}_\mu$

Let $\mathcal{N}$ be a label transition system, which represents a PN model, and $\delta : \mathcal{Z}_\mu \to 2^S$ is a firing sequence in the environment ($\mathcal{Z}_\mu \subseteq \mathcal{S}$).

$$\begin{aligned}
&[\top_\mu]^\delta_\mathcal{N} = \mathcal{S} \\
&[\bot_\mu]^\delta_\mathcal{N} = \phi \\
&[\mathcal{Z}_\mu]^\delta_\mathcal{N} = \delta(\mathcal{Z}_\mu) \\
&[\neg \mathcal{F}_\mu]^\delta_\mathcal{N} = \mathcal{S} \backslash [\mathcal{F}_\mu]^\delta_\mathcal{N} \\
&[\mathcal{F}_{\mu 1} \wedge \mathcal{F}_{\mu 2}]^\delta_\mathcal{N} = [\mathcal{F}_{\mu 1}]^\delta_\mathcal{N} \cap [\mathcal{F}_{\mu 2}]^\delta_\mathcal{N} \\
&[\mathcal{F}_{\mu 1} \vee \mathcal{F}_{\mu 2}]^\delta_\mathcal{N} = [\mathcal{F}_{\mu 1}]^\delta_\mathcal{N} \cup [\mathcal{F}_{\mu 2}]^\delta_\mathcal{N} \\
&[\mathcal{F}_{\mu 1} \to \mathcal{F}_{\mu 2}]^\delta_\mathcal{N} = (\mathcal{S} \backslash [\mathcal{F}_{\mu 1}]^\delta_\mathcal{N}) \cup [\mathcal{F}_{\mu 2}]^\delta_\mathcal{N} \\
&[\langle \beta \rangle \mathcal{F}_\mu]^\delta_\mathcal{N} = \{s \in \mathcal{S} | \exists_{t \in \mathcal{S}, \sigma \in [\beta]} s \xrightarrow{\sigma} t \wedge t \in [\mathcal{F}_\mu]^\delta_\mathcal{N}\} \\
&[[\beta] \mathcal{F}_\mu]^\delta_\mathcal{N} = \{s \in \mathcal{S} | \forall_{t \in \mathcal{S}, \sigma \in [\beta]} s \xrightarrow{\sigma} t \Rightarrow t \in [\mathcal{F}_\mu]^\delta_\mathcal{N}\} \\
&[\mu \mathcal{Z}_\mu . \mathcal{F}_\mu]^\delta_\mathcal{N} = \bigcap \{\mathcal{T}_\mu \subseteq \mathcal{S} | \mathcal{T}_\mu \supseteq [\varphi_\delta(\mathcal{T}_\mu)]\} \\
&[\nu \mathcal{Z}_\mu . \mathcal{F}_\mu]^\delta_\mathcal{N} = \bigcup \{\mathcal{T}_\mu \subseteq \mathcal{S} | \mathcal{T}_\mu \subseteq [\varphi_\delta(\mathcal{T}_\mu)]\},
\end{aligned}$$

where $\varphi_\delta : 2^\mathcal{S} \to 2^\mathcal{S}$ is a function defined as $\varphi_\delta(\mathcal{T}_\mu) = [\mathcal{F}_\mu]^{\delta[\mathcal{Z}_\mu := \mathcal{T}_\mu]}_\mathcal{N}$.

If $\varphi_\delta(M) \subseteq \varphi_\delta(N)$, where $M \subseteq N \subseteq \mathcal{S}$, then $\varphi_\delta$ is monotone operator on $\mathcal{S}$. If $\varphi_\delta(M) = M$, then $M$ is a fixed point of $\varphi_\delta$. By a well-known Knaster–Tarski theorem, we know that any monotone operator $\varphi_\delta$ on a set $\mathcal{S}$ has a least fixed point and a greatest fixed point within the ordering $(2^S, \subseteq)$.

The minimal fixed point is the intersection of prefixed points, $\bigcap \{M \subset \mathcal{S} | \varphi_\delta(M) \subseteq M\}$, and the maximal fixed point is the union of postfixed points, $\bigcup \{M \subset \mathcal{S} | M \subseteq \varphi_\delta(M)\}$. So we could extend our basic logic with a minimal fixed-point operator $\mu$, so that $\mu \mathcal{Z}_\mu . \mathcal{F}_\mu$ is a formula whose semantics is the least fixed point of $\varphi_\delta$, and similarly, a maximal fixed

point operator $\nu$, so that $\nu Z_{\mu}.\mathcal{F}_{\mu}$ is a formula whose semantics is the greatest fixed point of $\varphi_{\delta}$.

## 2.5 Self-adaptive multi-agent concurrent framework

SMACS as shown in Figure 2 is suitable for systems having complex behavior. To model such concurrent systems, formal techniques are being used for many decades, and fruitful results are being achieved by the researchers. In the proposed framework, an agent can perform multiple high-level tasks concurrently to complete the goal derives for the system or agent in a certain time span. These high-level concurrent tasks can be generated either by internal sensing of agents or by receiving updates by the support of external agents. An agent is self-directed, intelligent, and cooperative enough to collaborate with other agents to perform tasks in highly complicated environment. In this formal specification framework, we will use bottom-up approach for achieving true concurrency, as it is the best approach for modeling multi-agent-based systems to cover important concurrent aspects of agents. When a change in the environment will be observed, the Sensor channel intelligently sensed the request and notify to MAPE-K agents also known as internal-agents (Int-Agents) to complete the task. Each agent is composed of MAPE-K-based internal architecture. MAPE agents complete the request based on their knowledge and send back their decision to environment. These internal agents are directly connected to the CPN ML-based managed system and knowledge-based system updates for sending and receiving updates. Based on the results forwarded from the upper layer, these agents will act to complete the task and send updates to the environment through reactor channel for other agents.

All agents are also directly connected with each other for sharing updates, and each agent individually receive and share updates through Emitter and Consenter channels,

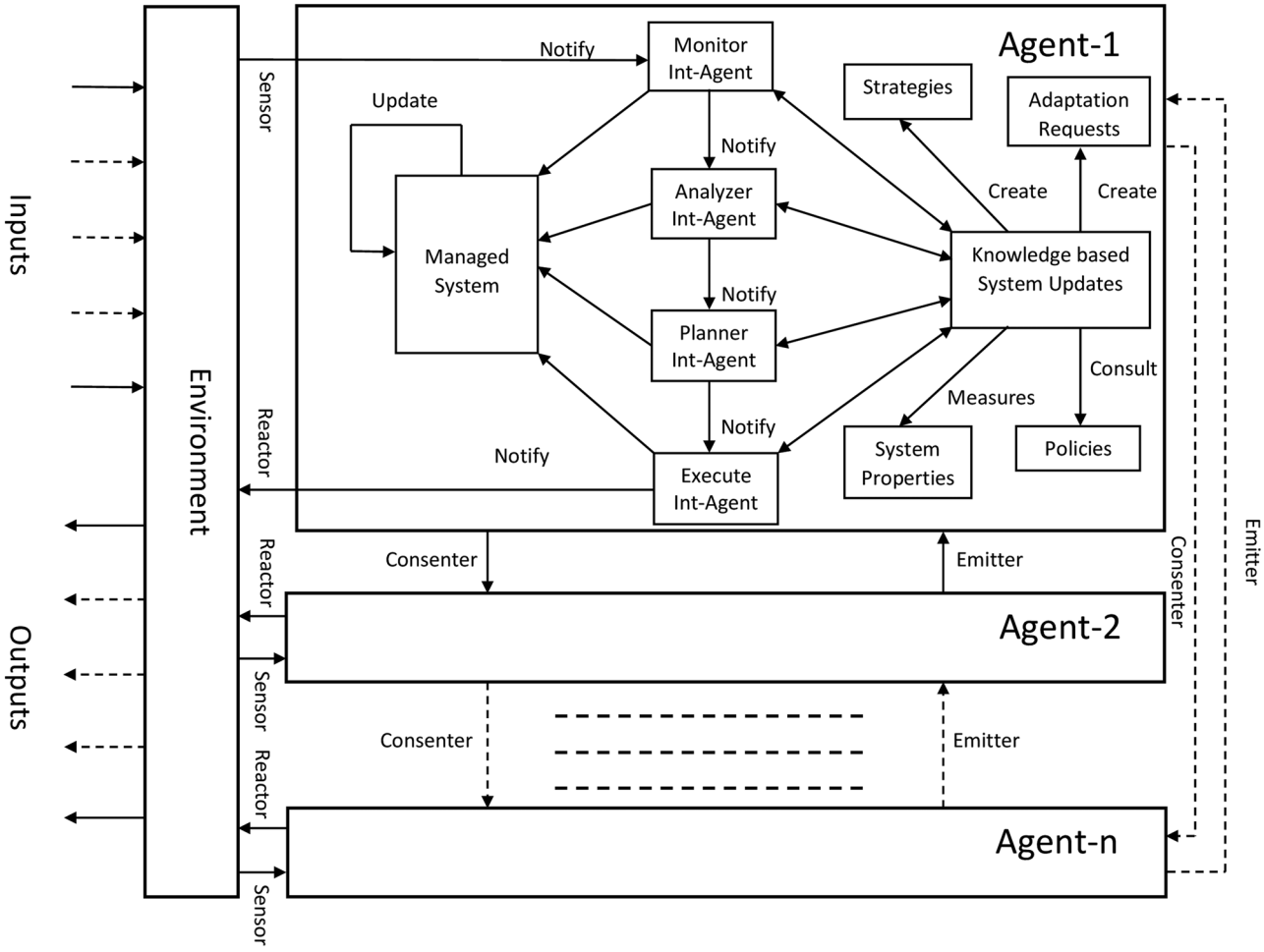


**Figure 2:** SMACS framework.

respectively. In some situations, there may be interaction between same type Int-Agents, i.e., monitor to monitor, analyzer to analyzer, planner to planner, and execute to execute. These internal-agents are directly connected to the knowledge based system updates for sending and receiving updated queries. Due this framework, $n$ numbers of agents can communicate concurrently to complete a task.

In the multi-agent environment, each internal agent of Agent-1 is directly connected with other agents' internal agents. Figure 3 represents the hierarchical structure of the SMACS framework, where all internal agents are grouped together from Sensor Int-Agent to Effector Int-Agent. As we know that in a MAS environment each agent's internal architecture is almost same, so we want to compact model, and the repeated portion of net will be compact to the hierarchical structure. All internal agents are grouped as a MAPE-K Agent.

## 3 Case study

To implement the SMACS framework, a SCL of any educational institution is chosen as a case study. The lab consists of limited number of computers. When a candidate wants to do some job, he generates a lab request (LR) through finger print device or by swapping a smart card for opening the lab. If there will be capacity inside the lab the door will open, otherwise a message will be printed on a screen attached outside the door that the lab is full. This working will be done through a sensor for checking seats availability, and updating status to the sensor attached with door opening lock. On the other hand, when a candidate will be able to enter in the lab, he will occupy a seat. As soon as he will sit on the seat, a message is forwarded by the sensor attached with seat to the sensor checking status of the system. If the system is off, it will become on. The status of a system will be checked after a specific time, and updates will be forwarded to all other sensors attached with it.

Concurrently, each agent will generate updates to the environment for other agents. Each agent is composed of MAPE-K-based architecture, and each agent can adapt the behavior either through changes in the environment or can forward updates to other agents. Similarly, other sensors (agents) will work concurrently for checking light status, fan status, temperature status, and humidity status. If there will be no candidate in the lab, all agents will send signal after a specific time to turning off the device attached with them. Printer is also a part of this Smart-Lab and will also be controlled through a sensor agent. This agent will receive updates from systems' agents for updating on or off status.

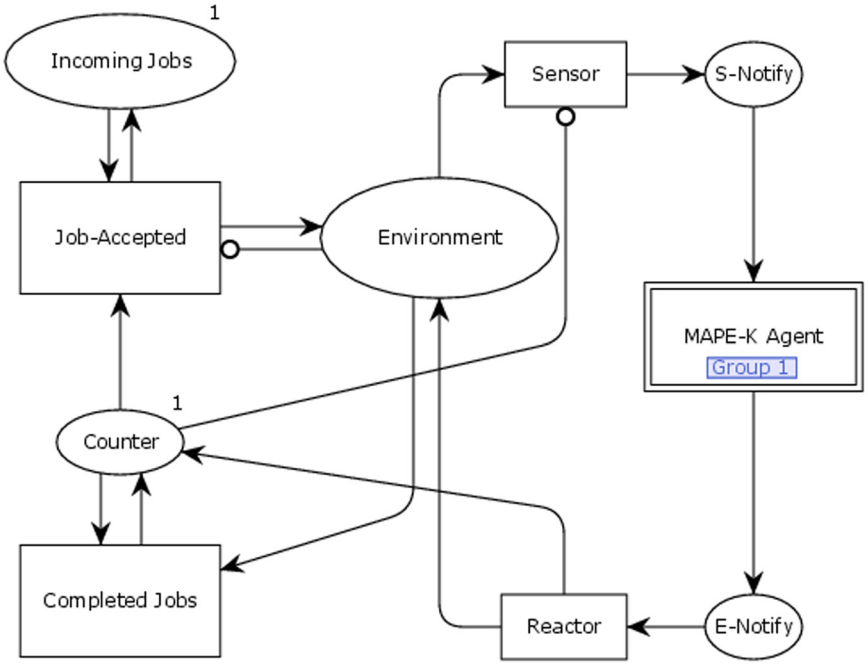


**Figure 3:** Hierarchical architecture of the SMACS framework.

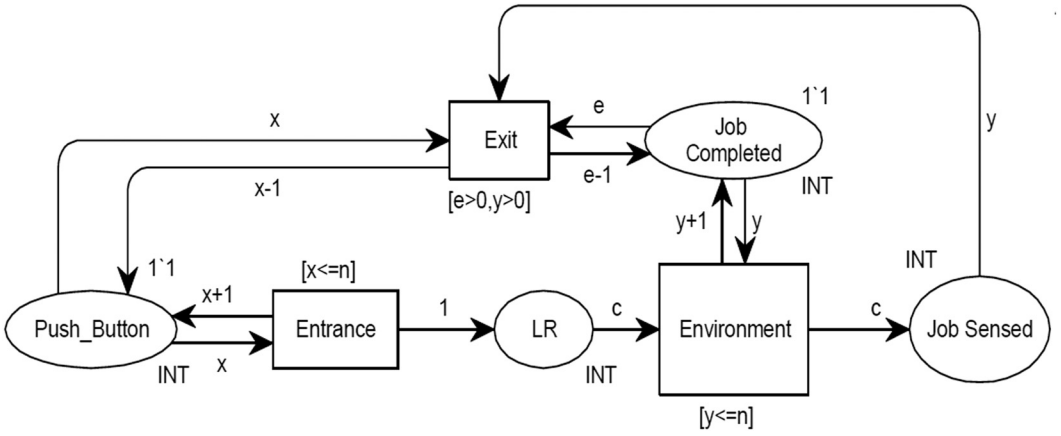


**Figure 4:** CPN module for entrance and exit.

CPN is a powerful modeling and analysis mathematical language with its graphical representations. By using CPN, we construct small modules, and finally, these small modules will combine to develop complete model through the bottom-up approach. Figure 4 represents the entrance and exit behavior of first agent. This agent adapts behavior on changing the capacity of SCL. ON/OFF is shown in Figure 5. Initially, the room temperature is set at 18°C, temperature will be increased 2° after every 60 min due to heat produced by the computer system. If temperature will rise up to 30°C or more, a signal is sent to AC controller agents and the AC switch will become ON. When the AC

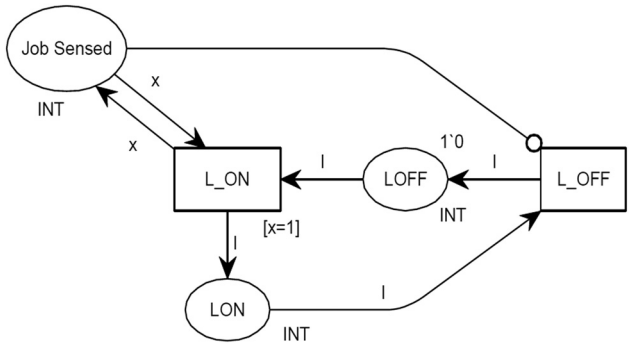


**Figure 5:** CPN module for light ON-OFF.

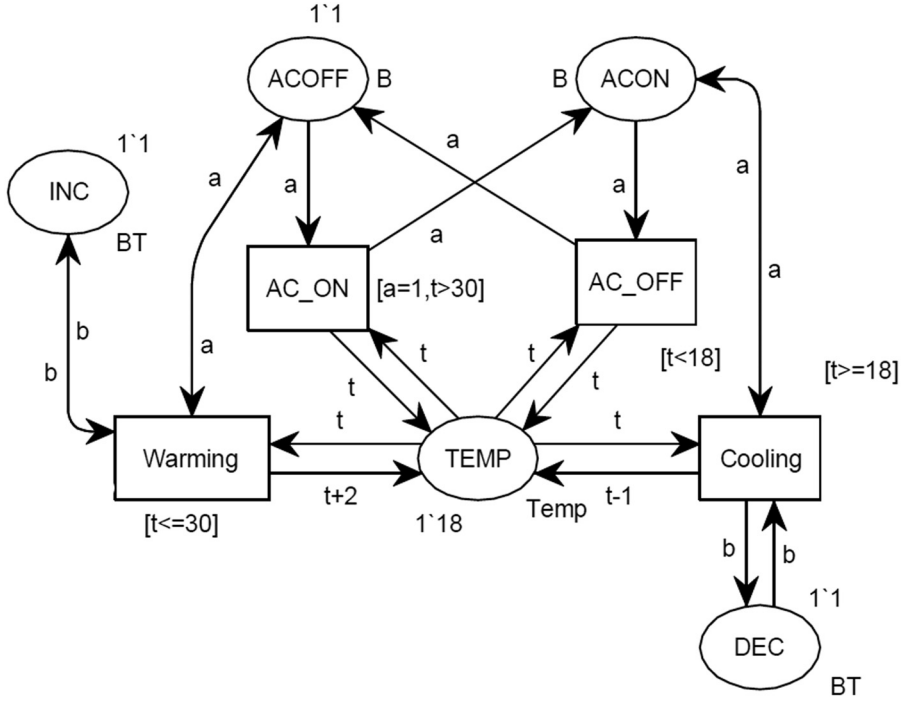


**Figure 6:** CPN module for AC controller.

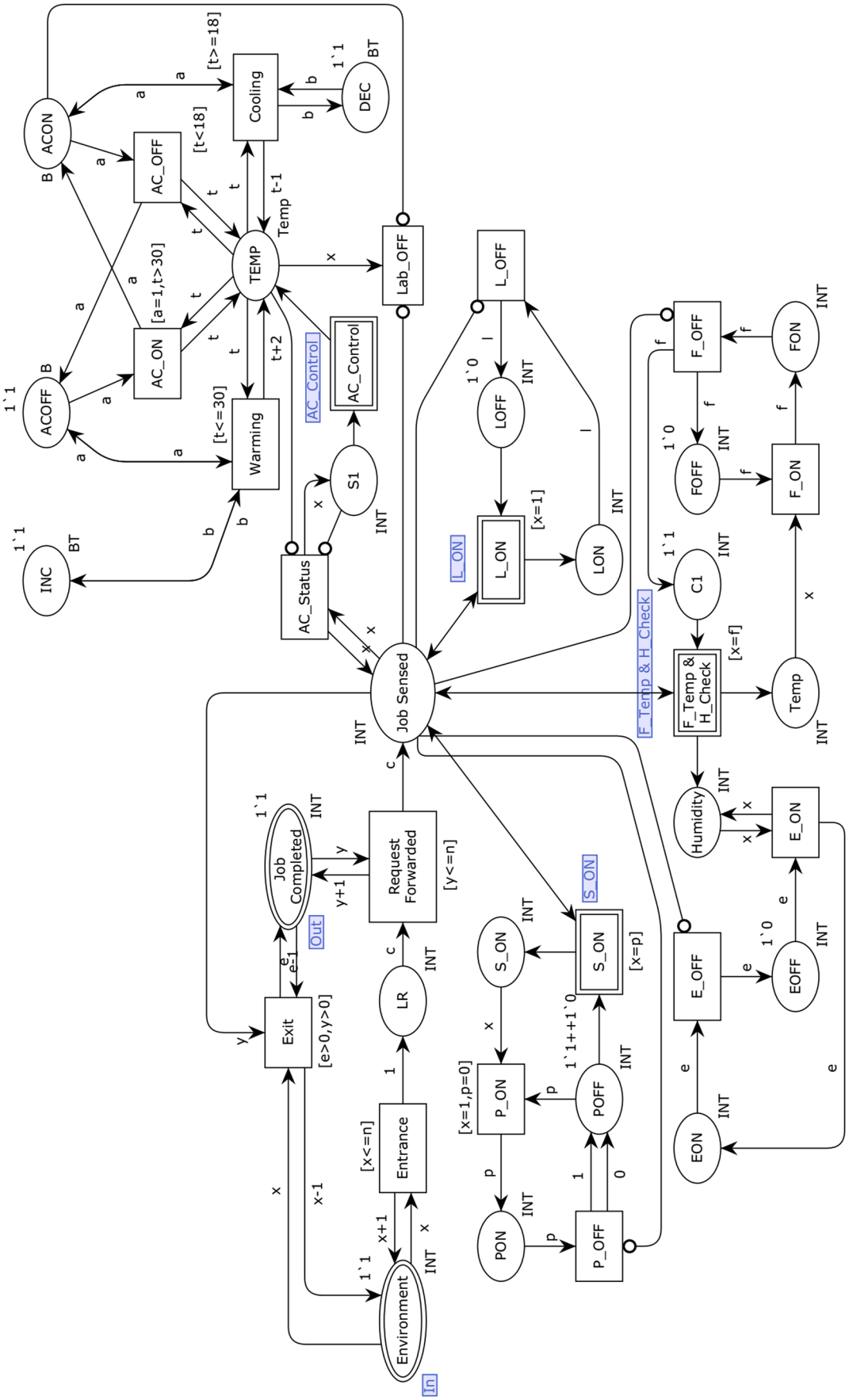


**Figure 7:** A meta-model of smart-lab using SMACS framework.

will start working, then after every 20 min, the temperature will become down to 1°. And if the temperature will become down to 18°C or less, the AC switch will become OFF by its agent. Similarly, the humidity measuring agent will control exhaust fan ON/OFF functions on the basis of humidity level of lab, when AC will be in the OFF mode. Finally, we will compose all modules to develop meta-model of Smart-Lab. When a candidate forwards his request through Entrance transition, a token is generated for LR place. Then environment transition may or may not be enabled according to the condition $y \leq n$, where $n$ is the maximum number of candidates and $y$ is associated with Job Completed place for counter. If the condition is true, the lab will open and candidate will occupy his seat, and at the same time, the system behavior will be updated. On the contrary, a signal of waiting is generated by the agent, which is monitoring lab capacity with its self-adaptive behavior. Initially, light is off, and a token is presented in L-OFF place, and at the same time, L_ON agent continuously check its status associated with Job Sensed place. If a seat is occupied by a candidate, the status of light will be changed to on and a token will be presented in the place Lab ON until all seats will become empty. This status will be updated by using inhibitor arc associated with Job Sensed place, i.e., the transition attached with the inhibitor arc will become fireable if the associated place will become empty. Similarly, all other agents associated with fan, systems, printer, temperature controller, and humidity controller will begin work as their status being updated by the environment. A timer of 15 min is set to all entities of the model with their idle status, and if during this interval there become no candidate in the lab, all running entities will be coming off their status through their agents. Temperature controller for air conditioner ON/OFF is shown in Figure 6. Initially, the room temperature is set at 18°C, temperature will be increased 2° after every 60 min due to heat produced by the computer system. If the temperature will rise up to 30°C or more, a signal is sent to AC controller agents and the AC switch will become ON. When the AC will start working, then after every 20 min, the temperature will become down to 1°C. And if the temperature decreases to 18°C or less, the AC switch will become OFF by its agent. Similarly, the humidity measuring agent will control exhaust fan ON/OFF functions on the basis of the humidity level of lab, when AC will be in the OFF mode. Finally, we will compose all modules to develop meta-model of Smart-Lab. Figure 7 represents the meta-model of SCL, where the transitions naming Warming, L_ON, S_ON, and F_Temp & H_Check are composed of MAPE-K internal agents.

# 4 Results and discussion

The results of the applying SMACS framework are briefly discussed in Sections 4.1 and 4.2. Liveness, safeness, and deadlock freedom properties are verified through the

**Table 1:** Modified $\mathcal{M}_\mu$ formulae for TAPA model checker to monitor lab atmosphere including liveness, safeness, and deadlock-freedom properties of SCL

| Property name | Modified formula for TAPA | Description |
|---|---|---|
| SCL1 | [Xi?][Capacity?](!⟨S_ON?⟩ true & ⟨L_ON?⟩ true & (⟨FH_C?⟩ true & (⟨F_ON?⟩ true \| ⟨Ex_ON?⟩ true)) & ⟨AC_Control?⟩ true) | Lab is open and empty, candidates can enter |
| SCL2 | [LAB_OFF!]((!⟨Xi?⟩ true & ⟨Empty?⟩ true & ⟨LAB_OFF?⟩ true ) \| (⟨P_OFF!⟩ true & ⟨S_OFF!⟩ true & ⟨FH_OFF!⟩ true & ⟨L_OFF!⟩ true & ⟨AC_OFF!⟩ true)) | Lab will close after leaving all candidates |
| SCL3 | [Xi?](⟨Ent?⟩ true & ⟨Capacity?⟩ true & ⟨R_F?⟩ true & ⟨Wait?⟩ false) | Lab has capacity to accommodate more candidates |
| SCL4 | [Xi?](⟨Wait?⟩ true & ⟨Ent?⟩ false & ⟨Capacity?⟩ false & ⟨R_F?⟩ false) | Lab is full and candidate can wait until a slot being empty |
| SCL5 | [Xi?][Capacity?][AC_Control?]((!⟨AC_ON?⟩ true (⟨Temp?⟩ true & !⟨AC_OFF?⟩ true) & ⟨E_OFF?⟩ \| (⟨Warm?⟩ true & ⟨Warm!⟩ true)) \| (⟨AC_OFF?⟩ (⟨Temp?⟩ true & !⟨AC_ON?⟩ true) \| (⟨Cool?⟩ true & ⟨Cool!⟩ true))) | AC control mechanism for Lab |
| **Liveness, safeness, and deadlock-freedom properties for SCL's internal agents** | | |
| SCL_L | !min Z. ((⟨Xi⟩ Z & ⟨Ent⟩ Z) & ⟨R_F⟩ Z \| ⟨S_ON⟩ Z \| ⟨L_ON⟩ Z \| ⟨FH_C⟩ Z \| ⟨AC_Control⟩ Z) \| max Y.([-tau]false) & (⟨tau⟩ true) & ([tau]Y) | Liveness property is checked for SCL |
| SCL_S | min Z. ([Ent]false \| [R_F]false \| [S_ON]false) \| [L_ON]false \| [FH_C]false \| [AC_Control]false \| ⟨ ∗ ⟩ Z | Safety property is checked for SCL |
| SCL_D | max Z. ([Xi][Ent](⟨R_F⟩ true & ⟨S_ON⟩ true ⟨L_ON⟩ true & ⟨FH_C⟩ true & ⟨AC_Control⟩ true)& [∗] Z ) | Deadlock freedom property is checked for SCL |

**Table 2:** State space analysis for CPN model of SCL

| **State space** | | |
|---|---|---|
| Node | 97,020 | |
| Arcs | 596,904 | |
| Secs | 2,808 | |
| Status | Full | |
| **SCC graph** | | |
| Node | 4 | |
| Arcs | 43,890 | |
| Secs | 11 | |
| **Boundedness properties** | | |
| **Best integer bounds** | | |
| **Place name** | **Upper** | **Lower** |
| SAS_PN'ACOFF 1 | 1 | 0 |
| SAS_PN'ACON 1 | 1 | 0 |
| SAS_PN'C1 1 | 1 | 0 |
| SAS_PN'DEC 1 | 1 | 1 |
| SAS_PN'EOFF 1 | 1 | 0 |
| SAS_PN'EON 1 | 1 | 0 |
| SAS_PN'Environment 1 | 1 | 1 |
| SAS_PN'FOFF 1 | 1 | 0 |
| SAS_PN'FON 1 | 1 | 0 |
| SAS_PN'Humidity 1 | 1 | 0 |
| SAS_PN'INC 1 | 1 | 1 |
| SAS_PN'Job Completed 1 | 1 | 1 |
| SAS_PN'Job Sensed 1 | 10 | 0 |
| SAS_PN'LOFF 1 | 1 | 0 |
| SAS_PN'LON 1 | 1 | 0 |
| SAS_PN'LR 1 | 10 | 0 |
| SAS_PN'POFF 1 | 2 | 0 |
| SAS_PN'PON 1 | 1 | 0 |
| SAS_PN'S1 1 | 1 | 0 |
| SAS_PN'S ON 1 | 1 | 0 |
| SAS_PN'TEMP 1 | 1 | 0 |
| SAS_PN'Temp 1 | 1 | 0 |
| **Home properties** | | |
| Home markings | 27,720 | [30,621, 36,282, 41, 942, 36,287, 25,154, ...] |
| **Liveness properties** | | |
| Dead markings | None | |
| Dead transition instances | None | |
| **Fairness properties** | | |
| Impartial transition instances | None | |
| Fair transition instances | None | |
| Just transition instances | None | |

TAPA model checker. Moreover, some properties are also discussed by applying the state-space analysis of the SCL model.

## 4.1 Verification of SCL

For the verification of all internal agents' working of the SCL, we apply $\mathcal{M}_\mu$ and TAPA model checker. All equipment of SCL including entrance/exit monitoring, computer/printer monitoring, lights/fans monitoring, atmosphere monitoring, and AC control monitoring cameras are working as internal agents. These agents are modeled through CPN and then verified through $\mathcal{M}_\mu$ with its corresponding TAPA model checker. The properties given in equations (3)–(8) show the lab capacity, lab on/off, lab's AC controller, liveness, safeness, and deadlock freedom. These properties show the formally correctness of the system. In formal methods, the system correctness lies in the verification of Assumption Guarantee paradigm for system's input/output functionality, liveness, and safety properties. The properties enlisted in Table 1 fall under these categories and therefore their correctness leads to the correctness of the system at large. Therefore, the formulation of these properties is being done in the modal $\mu$-calculus one of the formal methods. The purpose of this research work is to verify the soundness and completeness of the SMACS framework with the help of SCL case study.

$$[[Xi?](\langle \text{Ent?}\rangle \text{true} \wedge \langle \text{Capacity?}\rangle \text{true} \wedge \langle R\_F?\rangle \text{true} \wedge \langle \text{Wait?}\rangle \text{false})]_N^\delta \tag{3}$$

$$[[Xi](\langle \text{Wait?}\rangle \text{true} \wedge \langle \text{Ent?}\rangle \text{false} \wedge \langle \text{Capacity?}\rangle \text{false} \wedge \langle R\_F?\rangle \text{false})]_N^\delta \tag{4}$$

$$[[\text{LAB\_OFF!}](\neg\langle \text{Xi?}\rangle \text{true} \wedge \langle \text{Empty?}\rangle \text{true} \wedge \langle \text{LAB\_OFF?}\rangle \text{true}) \vee [\text{LAB\_OFF!}](\langle \text{P\_OFF!}\rangle \text{true} \wedge \langle \text{S\_OFF!}\rangle \text{true} \wedge \langle \text{FH\_OFF!}\rangle \text{true} \wedge \langle \text{L\_OFF!}\rangle \text{true} \wedge \langle \text{AC\_OFF!}\rangle \text{true})]_N^\delta \tag{5}$$

$$[\neg\mu\mathcal{Z}_\mu.\ ((\langle Xi\rangle\mathcal{Z}_\mu \wedge \langle \text{Ent}\rangle\mathcal{Z}_\mu) \wedge \langle R\_F\rangle\mathcal{Z}_\mu \vee \langle S\_\text{ON}\rangle\mathcal{Z}_\mu \vee \langle L\_\text{ON}\rangle\mathcal{Z}_\mu \vee \langle \text{FH\_C}\rangle \vee \langle \text{AC\_Control}\rangle\mathcal{Z}_\mu) \vee \mathcal{Y}_\mu([-\text{tau}]\text{false}) \wedge (\langle \text{tau}\rangle \text{true}) \wedge ([\text{tau}]\mathcal{Y}_\mu)]_N^\delta \tag{6}$$

$$[\mu\mathcal{Z}_\mu.\ ([\text{Ent}]\text{false} \vee [R\_F]\text{false} \vee [S\_\text{ON}]\text{false} \vee [L\_\text{ON}]\text{flase} \vee [\text{FH\_C}]\text{false} \vee [\text{AC\_Control}]\text{flase}) \vee \langle * \rangle\mathcal{Z}_\mu]_N^\delta \tag{7}$$

$$[\nu\mathcal{Z}_\mu.\ ([\text{Xi}][\text{Ent}](\langle R\_F\rangle \text{true} \wedge \langle S\_\text{ON}\rangle \text{true} \wedge \langle \text{L\_ON}\rangle \textit{true} \wedge \langle \text{FH\_C}\rangle \text{true} \wedge \langle \text{AC\_Control}\rangle \text{true}) \wedge [*]\mathcal{Z}_\mu]_N^\delta \tag{8}$$

These properties explained in Table 1 are written according to the TAPA model checker.

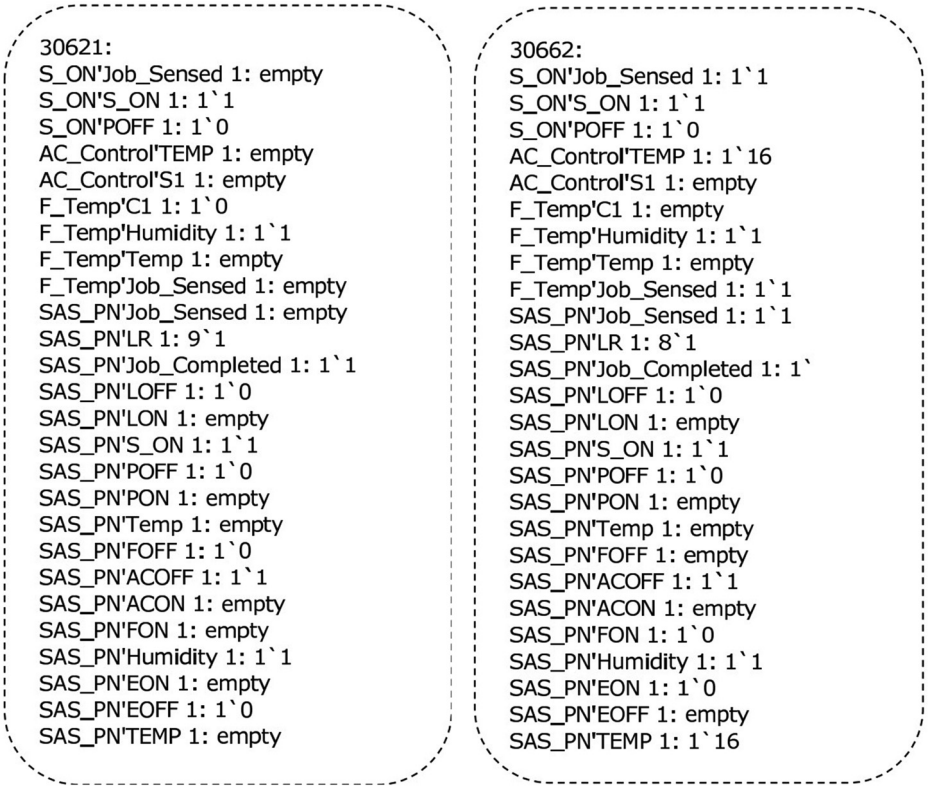


**Figure 8:** The details of node 3,0621 and 30,662 in the state space graph for SCL.

## 4.2 State space analysis for SCL

The graph size of the state space analysis of SCL is very huge, and it is impossible to show the whole graph. Table 2 represents that there are 97,020 nodes and 596,904 arcs, and it took about 47 min to generate the full state space report. Similarly, the strongly connected component (SCC) graph has four nodes and 43,890 arcs and took 11 s to generate this report. Four nodes of SCC graph mean that there are four strongly connected components to cover the all 97,020 nodes of the state space graph. Here, it is proved that the system will run in the infinite number of occurrences, and for termination of the system, we have to limiting the cycles. The best integer bound report represents the maximum and minimum integral value of any place, and the results show that no any place exists with the unbounded status. The best upper multiset bound shows that the maximum number of tokens exists in place during execution, and the best lower multiset bounds means all places have empty status other than initialized places. There are 27,720 markings represented as home marking, and the description of two home markings 30,621 and 30,662 is shown in Figure 8. The liveness property shows that there is no dead marking in the state space graph of the system and also no dead transition instances. A marking is dead if it does not have an outgoing arc. A dead transition instance means a transition is disabled to fire in all of its reachable markings. There are a number of live transitions exist in the graph. A partial state space graph shown in Figure 9 represents the description some arcs and some markings, e.g., $M_1$, $M_5$, and $M_{888}$, the arcs also describes through equations (9)–(11).

$$[6 : 3 \rightarrow 1\text{SAS\_PN'Exit } 1 : \{x = 2, y = 1, e = 2\}], \quad (9)$$

$$[12 : 5 \rightarrow 11\text{SAS\_PN'Entrance } 1 : \{x = 2\}], \quad (10)$$

$$[2286 : 314 \rightarrow 792\text{SAS\_PN'Request\_Forwarded } 1 : \{c = 1, y = 2\}]. \quad (11)$$

The details of three arcs represented in equations (9)–(11), e.g., in equation (11) the arc-2286 shows that a transition Request_Forward is fired from marking $M_{314}$ to marking $M_{792}$ when the values of variables are $c = 1$ & $y = 2$. As the system contains infinite occurrence sequence, so the fairness property shows that the system runs correctly and all transitions fire independently. A fairness property is imposed on the system that it fairly selects the process to be executed next. Technically, a fairness constraint is a condition on executions of the system model.

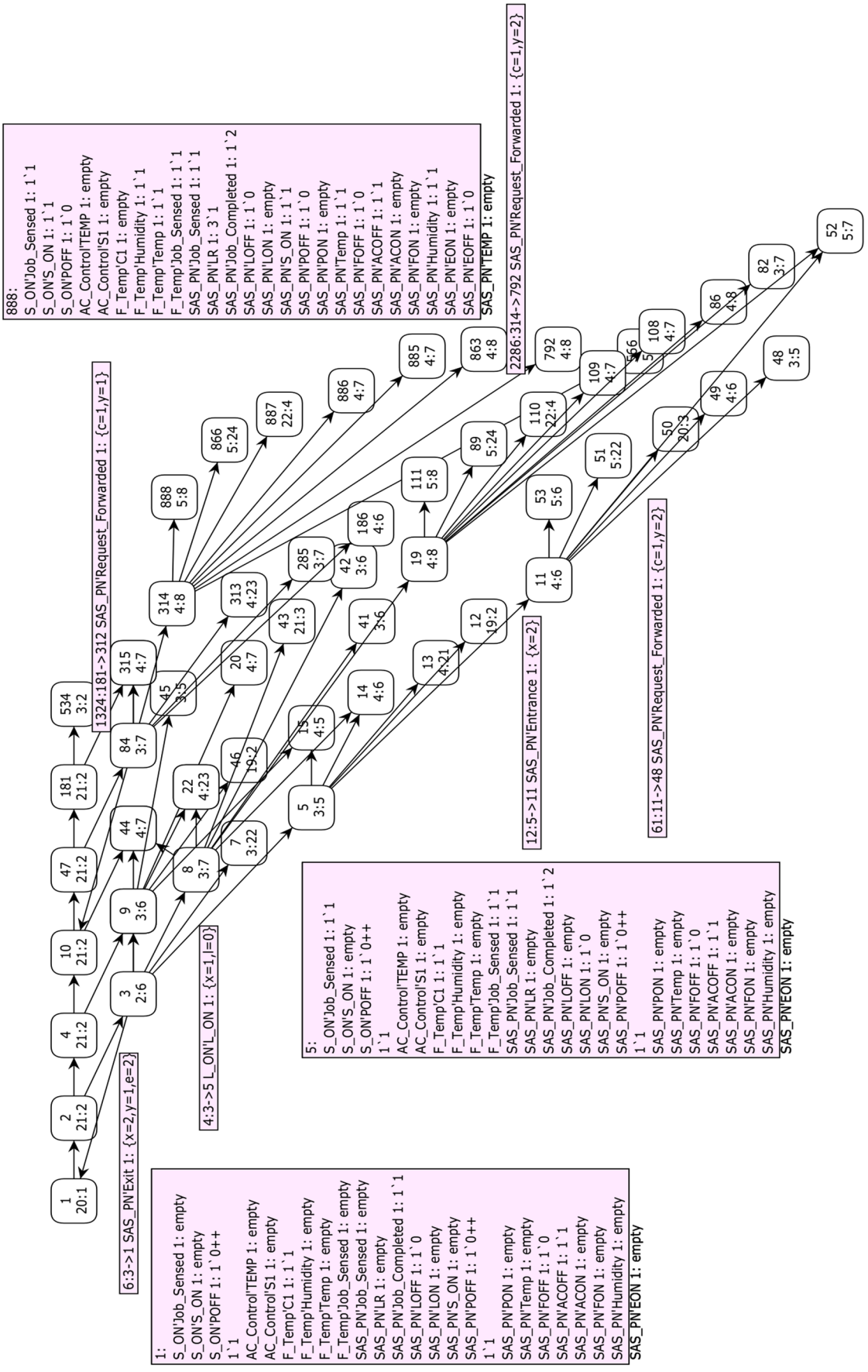


**Figure 9:** Partial state space graph for SCL.

## 5 Conclusions and future work

In this research, we tried to introduce a multi-agent-based framework for complex concurrent systems, where each agent is composed of MAPE-K-based internal structure. To ensure high level of reliability and accuracy of SMACS framework, we used a complex architecture of multi-agent-based SCL, as a case study. CPN is used as a modeling and verifying approach for SCL by using the SMACS framework, because CPN is more expressive than simple PNs to model such complex systems having concurrent behavior. Multi-agent-based and self-adaptive-based architectures are key features of our proposed work. Presently, these two fields having a key role for modeling such real-time complex architectures. It is verified that no deadlock occurs during the interaction between agents and internal agents. All agents interacted with each other through CPN ML-based managed system and knowledge-based updates. So due to any changes in the environment or any internal change of an individual agent, different agents adapted their behavior. It is formally verified that the proposed work is suitable to prove correctness of the system having complex concurrent behavior. It is also verified that there is no any dead marking, all transitions are live, and the net also hold safety property. The internal self-adaptive structure of SCL agents is verified through Modal $\mu$-calculus and then checked through the TAPA model checker. In our future work, we will try to add time stamp to verify the correctness of timed-based complex systems. We will also improve performance of the proposed architecture by using CPN-based statistical performance analysis tool and will try to use CPN monitors to analyze model based on statistical data. We also want to use temporal logic-based formal languages to handle the constraints of the proposed work with time variants in our future work.

**Funding information:** The authors state no funding involved.

**Author contributions:** All authors have accepted responsibility for the entire content of this manuscript and approved its submission.

**Conflict of interest:** Authors state no conflict of interest.

**Data availability statement:** Data sharing is not applicable to this article as no datasets were generated or analysed during the current study.

## References

[1] P. Sawyer, N. Bencomo, J. Whittle, E. Letier, and A. Finkelstein, "Requirements-aware systems: A research agenda for re for self-adaptive systems," In: *2010 18th IEEE International Requirements Engineering Conference* (*RE*), IEEE, 2010, pp. 95–103.

[2] M. Hinchey, *Requirements Engineering for Adaptive and Self-adaptive Systems*. 2018.

[3] J. Schmitt, M. Roth, R. Kiefhaber, F. Kluge, and T. Ungerer, "Realizing self-x properties by an automated planner," In: *Proceedings of the 8th ACM International Conference on Autonomic Computing*, ACM, 2011, pp. 185–186.

[4] G. Lilis and M. Kayal, "A secure and distributed message oriented middleware for smart building applications," *Autom. Constr.*, vol. 86, pp. 163–175, 2018.

[5] P. denHamer and T. Skramstad, "Autonomic service-oriented architecture for resilient complex systems," In: *2011 30th IEEE Symposium on Reliable Distributed Systems Workshops* (*SRDSW*), IEEE, 2011, pp. 62–66.

[6] H. Schmeck, C. Müller-Schloer, E. Čakar, M. Mnif, and U. Richter, "Adaptivity and self-organization in organic computing systems," *ACM Trans. Autonom. Adaptive Syst.* (*TAAS*), vol. 5, no. 3, p. 10, 2010.

[7] N. Villegas, G. Tamura, and H. Müller, "Architecting software systems for runtime self-adaptation: Concepts, models, and challenges," In: *Managing Trade-Offs in Adaptable Software Architectures*, Elsevier, 2017, pp. 17–43.

[8] C. Peper and D. Schneider, "Component engineering for adaptive ad-hoc systems," In: *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-managing Systems*, ACM, 2008, pp. 49–56.

[9] M. Gula and K. Žáková, "Proposal of component based architecture for internet of things: online laboratory case study," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 337–342, 2017.

[10] Y. Brun, G. D. M. Serugendo, C. Gacek, H. Giese, H. Kienle, M. Litoiu, et al., "Engineering self-adaptive systems through feedback loops," In: *Software Engineering for Self-adaptive Systems*, Springer, 2009, pp. 48–70.

[11] F. M. Favarò and J. H. Saleh, "Application of temporal logic for safety supervisory control and model-based hazard monitoring," *Reliabil. Eng. Syst. Safety*, vol. 169, pp. 166–178, 2018.

[12] A. O. de Sousa, C. I. Bezerra, R. M. Andrade, and J. M. Filho, "Quality evaluation of self-adaptive systems: Challenges and opportunities," In: *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, ACM, 2019, pp. 213–218.

[13] A. J. Ramirez, A. C. Jensen, and B. H. Cheng, "A taxonomy of uncertainty for dynamically adaptive systems," In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE Press, 2012, pp. 99–108.

[14] N. Esfahani and S. Malek, "Uncertainty in self-adaptive software systems," In: *Software Engineering for Self-Adaptive Systems II*, Springer, Berlin, Heidelberg, 2013, pp. 214–238.

[15] M. Krichen, "Improving formal verification and testing techniques for internet of things and smart cities," *Mobile Netw. Appl.*, pp. 1–12, 2019.

[16] B. H. Cheng, K. I. Eder, M. Gogolla, L. Grunske, M. Litoiu, H. A. Müller, et al., "Using models at runtime to address assurance for self-adaptive systems," In: *Models@run.time*, Springer, 2014, pp. 101–136.

[17] L. Castannneda, N. M. Villegas, and H. A. Müller, "Self-adaptive applications: On the development of personalized web-tasking systems," In: *Proceedings of the 9th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, ACM, 2014, pp. 49–54.

[18] H. Müller and N. Villegas, "Runtime evolution of highly dynamic software," In: *Evolving Software Systems*, Springer, 2014. pp. 229–264.

[19] N. Villegas, G. Tamura, H. Müller, L. Duchien, and R. Casallas, "Dynamico: A reference model for governing control objectives and context relevance in self-adaptive software systems," *Software Engineering for Self-Adaptive Systems II*, 2012.

[20] J. Cámara, R. de Lemos, N. Laranjeiro, R. Ventura, and M. Vieira, "Testing the robustness of controllers for self-adaptive systems," *J. Brazilian Comput. Soc.*, vol. 20, no. 1, p. 1, 2014.

[21] R. De Lemos, H. Giese, H. Müller, M. Shaw, J. Andersson, L. Baresi, et al., "Software engineering for self-adaptive systems: A second research roadmap," In: *Dagstuhl Seminar Proceedings*, Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2011.

[22] G. Weiss, K. Becker, B. Kamphausen, A. Radermacher, and S. Gerard, "Model-driven development of self-describing components for self-adaptive distributed embedded systems," In: *2011 37th EUROMICRO Conference on Software Engineering and Advanced Applications* (*SEAA*), IEEE, 2011, pp. 477–484.

[23] H. Müller, M. Pezzè, and M. Shaw, "Visibility of control in adaptive systems," In: *Proceedings of the 2nd International Workshop on Ultra-large-scale Software-intensive Systems*, ACM, 2008, pp. 23–26.

[24] S. Dobson, S. Denazis, A. Fernández, D. Gaíti, E. Gelenbe, F. Massacci, et al., "A survey of autonomic communications," *ACM Trans. Autonom. Adaptive Syst.* (*TAAS*), vol. 1, no. 2, pp. 223–259, 2006.

[25] K. Jensen. *Coloured Petri Nets: Basic Concepts, Analysis Methods and Practicaluse*, 2013, https://doi.org/10.1186/s40294-016-0015-x.

[26] M. A. Niazi. *Introduction to the modeling and analysis of complex systems: a review*, 2016, https://doi.org/10.1186/s40294-016-0015-x.

[27] K. Jensen, L. M. Kristensen, and L. Wells, "Coloured petri nets and cpn tools for modelling and validation of concurrent systems," *Int. J. Softw. Tools Technol. Transf.*, vol. 9, no. 3–4, pp. 213–254, 2007.

[28] F. Liu, M. Heiner, and D. Gilbert, "Coloured petri nets for multilevel, multiscale and multidimensional modelling of biological systems," *Briefings Bioinform.*, vol. 20, no. 3, pp. 877–886, 2017.

[29] A. Qasim and S. A. R. Kazmi, "Mape-k interfaces for formal modeling of real-time self-adaptive multi-agent systems," *IEEE Access*, vol. 4, pp. 4946–4958, 2016.

[30] W. Wusheng, L. Weiping, W. Zhonghai, and Z. Zhichao, "Petri net-based context-aware service system modelling: an overview," In: *2014 International Conference on Service Sciences* (*ICSS*), IEEE, 2014, pp. 60–65.

[31] F. D. Macías-Escrivá, R. Haber, R. delToro, and V. Hernandez, "Self-adaptive systems: A survey of current approaches, research challenges and applications," *Expert Systems with Applications*, vol. 40, no. 18, pp. 7267–7279, 2013.

[32] D. Weyns and M. Georgeff, "Self-adaptation using multiagent systems," *IEEE Software*, vol. 27, no. 1, 2010.

[33] M. I. Fakhir and S. A. R. Kazmi, "Formal specification and verification of self-adaptive concurrent systems," *IEEE Access*, vol. 6, pp. 34790–34803, 2018.

[34] Y. Abuseta and K. Swesi, *Design Patterns for Self Adaptive Systems Engineering*, 2015, arXiv: http://arXiv.org/abs/arXiv:1508.01330.

[35] W. Kröger and E. Zio, "Vulnerable Systems," Springer, London, 2011.

[36] I. Fakhir, S. A. R. Kazmi, A. Qasim, and I. Rafique, "Concurrency in intuitionistic linear-time $\mu$-calculus: A case study of manufacturing system," *Indian J. Sci. Technol.*, vol. 9, no. 6, pp. 1–7, 2016.

[37] S. A. R. Kazmi and W. H. Zhang, "Compositional reasoning in intuitionistic linear time mu-calculus,", *J. Softw.*, vol. 20, no. 8, pp. 2026–2036, 2009.

[38] M. Garcia-Constantino, A. Konios, and C. Nugent "Modelling activities of daily living with petri nets," In: *2018 IEEE International Conference on Pervasive Computing and Communications Workshops* (*PerCom Workshops*), IEEE, 2018, pp. 866–871.

[39] A. Kayes, W. Rahayu, T. Dillon, S. Mahbub, E. Pardede, and E. Chang, "Dynamic transitions of states for context-sensitive access control decision," In: *International Conference on Web Information Systems Engineering*, Springer, 2018, pp. 127–142.

[40] F.-S. Hsieh, "A hybrid and scalable multi-agent approach for patient scheduling based on petri net models," *Appl. Intell.*, vol. 47, no. 4, pp. 1068–1086, 2017.

[41] A. Qasim, A. Kazmi, and I. Fakhir, "Formal specification and verification of real-time multi-agent system using timed arc petri nets," *Adv. Electr. Comput. Eng.*, vol. 15, no. 3, pp. 73–8, 2015.

[42] M. Wooldridge and N. R. Jennings, "Intelligent agents: Theory and practice," *Knowl. Eng. Rev.*, vol. 10, no. 2, pp. 115–152, 1995.

[43] D. Weyns, "Software engineering of self-adaptive systems," In: *Handbook of Software Engineering*, Springer, 2019, pp. 399–443.

[44] A. Qasim and S. A. R. Kazmi, "Mape-k interfaces for formal modeling of real-time self-adaptive multi-agent systems," *IEEE Access*, vol. 4, pp. 4946–4958, 2016.

[45] D. Weyns, M. U. Iftikhar, S. Malek, and J. Andersson, "Claims and supporting evidence for self-adaptive systems: A literature study," In: *Proceedings of the 7th International Symposium on Software Engineering for Adaptive and Self-Managing Systems*, IEEE Press, 2012, pp. 89–98.

[46] M. Salehie and L. Tahvildari, "Towards a goal-driven approach to action selection in self-adaptive software," *Software: Practice and Experience*, vol. 42, no. 2, pp. 211–233, 2012.

[47] M. Hennessy and R. Milner, "Algebraic laws for nondeterminism and concurrency," *J. ACM* (*JACM*), vol. 32, no. 1, pp. 137–161, 1985.

[48] P. Blackburn, J. F. van Benthem, and F. Wolter, *Handbook of Modal Logic*, vol. 3, Elsevier, UK, 2006.