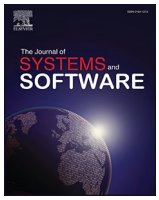
The Journal of Systems and Software 210 (2024) 111958

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss



Modeling and safety analysis for collaborative safety-critical systems using hierarchical colored Petri nets

Nazakat Ali [a,*], Sasikumar Punnekkat [a], Abdul Rauf [b]

[a] School of Innovation, Design and Technology, Mälardalen University, Västerås, Sweden
[b] Knightec AB, Västerås, Sweden

ARTICLE INFO

Editor: W. Eric Wong

Keywords:
Safety-critical
Petri nets
Safety analysis
Colored Petri-nets

ABSTRACT

Context: Collaborative systems enable multiple independent systems to work together towards a common goal. These systems can include both human-system and system-system interactions and can be found in a variety of settings, including smart manufacturing, smart transportation, and healthcare. Safety is an important consideration for collaborative systems because one system's failure can significantly impact the overall system performance and adversely affect other systems, humans or the environment.
Goal: Fail-safe mechanisms for safety-critical systems are designed to bring the system to a safe state in case of a failure in the sensors or actuators. However, a collaborative safety-critical system must do better and be safe-operational, for e.g., a failure of one of the members in a platoon of vehicles in the middle of a highway is not acceptable. Thus, failures must be compensated, and compliance with safety constraints must be ensured even under faults or failures of constituent systems.
Method: In this paper, we model and analyze safety for collaborative safety-critical systems using hierarchical Coloured Petri nets (CPN). We used an automated Human Rescue Robot System (HRRS) as a case study, modeled it using hierarchical CPN, and injected some specified failures to check and confirm the safe behavior in case of unexpected scenarios.
Results: The system behavior was observed after injecting three types of failures in constituent systems, and then safety mechanisms were applied to mitigate the effect of these failures. After applying safety mechanisms, the HRRS system's overall behavior was again observed both in terms of verification and validation, and the simulated results show that all the identified failures were mitigated and HRRS completed its mission.
Conclusion: It was found that the approach based on formal methods (CPN modeling) can be used for the safety analysis, modeling, validation, and verification of collaborative safety-critical systems like HRRS. The hierarchical CPN provides a rigorous way of modeling to implement complex collaborative systems.

1. Introduction

In the era of industry 4.0, the collaboration between/among systems is an important aspect where a common mission can be achieved through the collaboration of systems, and even collaboration between those systems with humans enables the necessary abstraction (Lee et al., 2015). This concept led to the emergence of collaborative systems, defined as "jointly acting and sharing information, resource, and responsibilities in order to achieve a common goal" (Maier, 1998). The motivation for using collaborative systems is to achieve capabilities that cannot be achieved by a single system alone. A capability can be, for example, a collaborative or complex service delivered to the system's end users or to other subsystems/systems. For instance, several vehicles form a group in the platoon driving system and drive with a narrow inter-vehicle gap to increase traffic flow and fuel reduction. Therefore, collaboration is one of the corner stones of the modern systems. Furthermore, collaborative systems provide unprecedented capabilities and opportunities due to growing intelligence, autonomy, and interconnection. However, safety concern is among the main challenges when designing and deploying collaborative safety-critical systems. For instance, if a leader vehicle in an autonomous platoon driving fails to communicate with other member vehicles, then the critical commands such as emergency stop from the leader vehicle become unavailable, and driving with a short gap would not be safe anymore (Ali et al., 2021).

* Corresponding author.
E-mail address: nazakat.ali@mdu.se (N. Ali).

https://doi.org/10.1016/j.jss.2024.111958
Received 17 May 2023; Received in revised form 22 November 2023; Accepted 2 January 2024
Available online 3 January 2024

Emergent behavior in a collaborative system can lead to overall system failure if it is not properly designed, verified, or implemented (Raman and Murugesan, 2022). Failures in collaborative systems can have multiple causes, including constituent system failures, component failures, interoperability issues, human errors, cyber-attacks, and environmental factors. From a developer's perspective, factors such as system design flaws, lack of adequate verification, and unmanaged system complexity can lead to the above failure causes. Identifying and mitigating these potential failures is important to ensure the collaborative systems' reliability and safety. Researchers (Hussain et al., 2022; Gualtieri et al., 2021) are working on different aspects to fix the problems related to collaboration in safety-critical systems to ensure that the collaborative systems are safe before entering the deployment phase and work correctly in all identified scenarios. Formal methods are often recommended for the modeling and analysis of safety-critical systems to bring acceptable confidence in such complex systems. Therefore, formal methods such as Colored Petri Nets (CPN) are used for modeling both functional specifications and interaction design to ensure the collaborative systems meet all specified requirements.

Formal methods are a set of mathematical techniques used to formally specify, design, verify, and validate safety-critical systems. It can be used to ensure that a system meets its safety requirements and does not contain any faults that could lead to unsafe behavior (Liu, 2014). In contrast to traditional system design techniques that focus on extensive testing to verify the system behavior but can only draw partial conclusions, formal methods provide further insurance as they only accept systems that have been proved correctly (Bowen and Stavridou, 1993). CPN is one of the formal specification languages (Peterson, 1977) that enables us to model and analyze safety-critical systems, especially asynchronous, distributed, concurrent, non-deterministic, and parallel systems.

This paper aims to model, validate and verify collaborative safety-critical systems using hierarchical CPN. Hierarchical CPN is used for a variety of reasons, such as: it allows for the modeling and analysis of complex systems at different levels of abstraction; It is used to model and verify collaborative systems, which are composed of multiple interacting systems; By using the hierarchical structure of hierarchical CPN, it is possible to divide the system into smaller and more manageable subsystems, which can be modeled and verified individually (Jensen and Kristensen, 2015); it is a discrete event modeling language that combines basic Petri nets with the functional programming language CPN ML which is based on standard ML; CPN uses the graphical representation to describe the model simply and intuitively which enables us to better understand the requirements specification of the system compared with the textual descriptions; it also simulates and executes the model for system verification. The hierarchical system modeling provides a simplified net that can give a broad overview of the system. It also allows a system module to have submodules and reuse submodules in various parts of the model. The ability of the CPN to verify the system model formally is particularly important for safety-critical systems. Therefore, it has been used in modeling and safety analysis of safety-critical systems (Hu et al., 2017; Wu, 2014; Song and Schnieder, 2018; Wu and Zheng, 2018). Colored Timed Petri Net (CTPN) (Kuo and Huang, 2000) is a variant of CPN where the color attributes manage large systems that have many similar or redundant logical structures. Using a CPN variant without timing properties can be advantageous for simpler systems without significant time-sensitive behaviors. It can lead to faster modeling and analysis efforts. However, it comes at the cost of limited expressiveness and accuracy for systems that do involve timing constraints. CTPNs are well-suited for systems where timing and synchronization aspects are critical, whereas hierarchical CPNs are favored when dealing with large and complex systems that benefit from a structured and modular modeling approach (Li et al., 2016). When dealing with large and complex systems, hierarchical CPNs can be more scalable and maintainable compared to CTPNs. The hierarchical approach allows for easier management of system complexity as the system grows. Since we aim to model a complex collaborative system therefore we opted for hierarchical CPNs over other CPN variants.

Collaborative systems are distributed systems that work jointly to perform a higher-level complex task that a single constituent system cannot perform alone. Collaborative systems should be resilient by design since they are used in critical applications and should work even if some of the constituent systems fail. Different from most safety analysis techniques for systems that perform their duty independently and do not collaborate with each other to complete a common mission, our approach presents modeling and safety analysis for collaborative safety-critical systems using hierarchical CPN. The hierarchical CPN allows one to verify some properties of the underlying system by formal analysis independent of the simulation, which only indicates the presence of faults in the model instead of asserting the absence of faults. It also supports modularity in modeling the system using CPN to manage large systems.

The main problem in a collaborative system is that when one constituent system fails the collaborative system can not achieve its common goal. Therefore, collaborative systems should be modelled, verified, and validated carefully. In this study, we modelled a collaborative safety-critical system and selected hierarchical CPN for modelling. Verifying and validating collaborative systems at the system level is another challenge that makes it difficult to ensure safety (Honour, 2013). We incorporated three failures at the system level in a collaborative system and verified the systeḿs safe behaviour in the presence of these failures by designing a safety mechanism for each injected failure. We verified the safe behaviour of a collaborative system using boundedness, deadlocks and dead transitions. We also validated the system to ensure that the system is free of livelocks, free of self-loop, and full state space is computed which in turn tells whether all variable states are involved in the simulation. From the verification and validation results, it was confirmed that the collaborative system achieves its common goal safely and the injected failures were acceptably mitigated.

In summary, we make the following contributions:

- We have proposed a unique approach to carry out simulation-based safety verification and validation of collaborative systems. This approach models, verifies, and validates a collaborative system using hierarchical CPN to ensure that the collaborative system achieves its common mission under hazardous scenarios.
- The proposed approach is demonstrated in a concrete use case of human rescue robot system, which can help researchers and practitioners to model, verify and validate complex collaborative systems. The boundedness, deadlocks and dead transition safety properties were used to verify the system while livelocks, self-loop, and state space were used to validate the system.

The research procedure is divided into following steps:

1) We used the human rescue robot system (HRRS) as a case study, modeled it using hierarchical CPN, and introduced some simulated failures to demonstrate its safe behavior.
2) In modeling HRRS using hierarchical CPN, three types of failures (representative of the sense, compute and actuate phases of control applications) are incorporated, i.e.:
   a. *Sensor Failure*: Failure due to the problem in sensing the environment that hinders the task of sensing objects of interest.
   b. *Navigation Failure*: Failures during navigating to the target location, e.g., slippery ground.
   c. *Object Clearance Failure*: Failures in clearing the area due to insufficient actuating mechanism on the robot.
3) After injecting faults that can cause the above failures, the system behavior was observed, and a state space graph was generated to check whether the hazardous states were reachable or dead.
4) Subsequently, safety mechanisms were defined for each failure mode, and the model was verified and validated against different

safety property requirements. The verification and validation results show that respective failures were acceptably mitigated, and HRRS completed its mission successfully.

The remainder of this paper is structured as follows. Section 2 represents the related work. Section 3 gives an overview of the background. Section 4 describes the procedure steps of our proposed approach. Section 4 describes the system modeling with CPN. Section 5 describes the simulation configuration of the CPN model. Section 6 defines the safety mechanism for injected failures. Section 7 discusses the CPN model verification and validation, and Section 8 concludes this paper.

2. Related work

Hu et al. (2017) proposed a Colored Petri Net (CPN) based hierarchical control structure to identify hazards related to the railway system and verify the model's consistency with the actual system. The proposed method is the combination of CPN and System-Theoretic Process Analysis (STPA), which is named formalSTPA. The formalSTPA identifies hazards by obtaining system hazards that may potentially lead to a mishap. It also identifies safety constraints to prevent hazards from occurring. After identifying safety constraints, a preliminary control structure is designed to fulfill requirements, and the control structure should be improved over time. The authors transformed the control structure model into the CPN model, which describes system behavior and the system's internal interactions. After obtaining the CPN model, ASK-CTL was used to verify the CPN model. Along with the CPN model, the state-space analysis was also used to identify the cause of hazards. Lastly, safety requirements were proposed to ensure system safety.

Gonçalves et al. (2017) presented a formalized safety assessment model to provide evidence of the safety and reliability of unmanned aerial vehicles (UAV). The proposed model shows the real dynamics of the decision-making process, including failure conditions. The authors made some assumptions for the implementation of the model. The assessment model includes four steps, i.e., identification of UAV flight phase, identification of failure conditions leading to most hazardous events, evaluation of failures in each flight phase, information to flight control, and decision making from control operators. In this study, the authors intended to show how the UAV enters the most feared events and the UAV ability to react after being in a fault situation. The results show that it was possible to identify and define safety-critical areas and corrective actions to prevent faults.

Wang et al. (2016) proposed a hazard identification method by combining STPA and STPA Based on the Formalization Method (BMF) called BMF-STPA. This approach combines the STPA hazard analysis technique with the formalization of CPN in order to determine system control structure models, identify safety hazards, and generate a hazard log. The main features of this BMF-SPTA approach include the combination of STPA with a formal modeling technique to generate an integrated hazard log. This approach can effectively determine interactive factors among external and internal interfaces. The BMF-STPA based approach also can conduct a hazard analysis and determine causes that lead to mishaps. The authors concluded that the BMF-STPA approach could model the system behavior through CPN to verify that safety behavior would be possible. Another reason for CPN was also to trace the hazards in the system through state space.

Song et al. (2017) used CPN as part of the verification process, which helps evaluate functional safety along with performance assessment using parameters from formal models developed for Train to Train Distance Measurement System (TTDMS). The authors made three main contributions. Firstly, they proposed a formalized TTDMS model and validated its correctness using state space analysis and simulation-based verification. Secondly, the corresponding checking queries were developed for the purpose of functional safety verification as well as to evaluate system performance by applying parameters in the formal models. Lastly, the authors estimated the reliability of a functional prototype TTDMS based on the processes which can be used during development stages, with both formal verifications and simulations being performed simultaneously for better results. The authors concluded that applying their process is easier than executable code or mathematical methods when evaluating and verifying systems due to improved readability and reliability compared to other approaches.

Akhtar et al. (2019) proposed a novel formal system for specification, analysis, design, modeling, and verification for a multi-agent safety-critical system. The proposed system is specified, designed, and analyzed using Gaia multi-agent methodology (Honour, 2013). The authors specified model-based agent roles along with regular expression, liveness properties, first-order predicate, and calculus-based safety properties that must also be correct, complete, consistent, and unambiguous. Finally, hierarchical CPN was used to model and verify the designed systems before implementation. The hierarchical CPN was constructed on two abstraction levels to achieve modularity.

Zhang et al. (2022) proposed a formal approach to quantitatively evaluate the operational efficiencies of the Centralized Traffic Control (CTC) system with respect to different safety control schemes. The proposed approach adopts stochastic CPN to describe the CTC system model, and evaluates its efficiency based on data collected during simulation. The proposed approach was exemplified by studying how prohibiting passenger trains from passing freight trains between two adjacent stations affects the CTC system's overall performance. Results showed that the proposed approach is an effective way to measure such systems' operations performances. The authors also mentioned that their approach could help in deciding which safety control scheme is most effective at improving operational efficiency while maintaining safe operations.

Summary: From the investigation of related work, we learned that the use of CPN in modelling and verifying safety-critical systems is evident. The authors Hu et al. (2017) and Wang et al. (2016) have combined STPA with the CPN. The integration of the CPN model, which replaced the original control structure model in STPA, significantly enhanced the systematic capabilities for modelling various features. Some authors have (Gonçalves et al., 2017; Zhang et al., 2022) used CPN for safety assessment process modelling in UAV case study and railway domain respectively. The obtained results enabled the identification and definition of critical areas as well as the formulation of corrective measures that will lead to an acceptable level of risk for the regulatory authority. However, the verification and validation were not considered. The verification and validation for a TTDMS were considered in (Song et al., 2017), however, the applied case study was not a collaborative system. In another study (Akhtar et al., 2019), a multi-agent system was modelled with CPN, but verification and validation were not considered. Safety professionals are using CPN for modelling and safety analysis for safety-critical systems. However, to the best of our knowledge, there is no study where a collaborative system is modelled with hierarchical CPN, verified, and validated at the system level while injecting failures.

3. Background

3.1. Formal definition of CPN

CPN is a high-level Petri net that allows the tokens to have a data value attached to it, called the token color. Formally, a CPN model $N = (\Sigma, P, T, I, O, C, G, V, M0)$ (Jensen and Kristensen, 2015; Valadares et al., 2021) where:

1) Σ represents a finite set of colors which are nonempty type;
2) $P = \{p1, p2, .., pm\}$ is a finite set of places;
3) $T = \{t_1, t_2, .., t_n\}$ is a finite set of transitions such that $P \cap T = \varnothing$;
4) I denotes an input function such that $P \times T \rightarrow N$ defines directed arcs from P to T, where N is a set of nonnegative integers;
5) denotes the output function such that $T \times P \rightarrow N$ is an output function that defines directed arcs from T to P;

6) C denotes color functions which are defined from P into Σ;
7) G is a guard function that is defined from set T into the expressions such that $[\forall t \in T$ *that is* $Type(G(t)) = Bool \wedge Type\ (Var\ (G(t))) \subseteq \Sigma]$;
8) V is the finite set of variables $v \in V$ of colors $c \in C$. Arc expressions and guards contain variables $v \in V$ of the suitable types;
9) $M0$ specifies the initial placement color of tokens on the places of the model. The $M0$ is the initial marking defined on P: $M0(p) \in Bag(C(p))$, $\forall p \in P$. The input and output functions are defined on $Bag(C(t))$.

Hierarchical CPN (Peterson, 1977) supports modularity in modeling the system using CPN to manage large systems. The Hierarchical CPN modules have a specific type of transition called substitution transitions and a specific type of place called the port place. The substitution transition substitutes a detailed module, while the port place acts as a buffer to send or receive tokens among various modules through substitution transitions.

A CPN module is a 4-tuple $CPN_{module} = (CPN, T_{sub}, P_{port}, PT)$ (An et al., 2018) where:

1. $CPN = (\Sigma, P, T, I, O, C, G, V, M0)$ is a CPN model without hierarchical properties
2. $T_{sub} \subseteq T$ represents a set of substitution transitions;
3. $P_{port} \subseteq P$ denotes a set of port places and
4. PT: $P_{port} \rightarrow$ (*In, Out, In/Out*) denotes a port-type function that is responsible for assigning a port type to each port place.

A hierarchical CPN can be expressed as a 4-tuple $M_{Hierarchal} = (S, SM, PS, FS)$ *where*:

1. S is a finite set of CPN modules. For each $CPN_{module} = (CPN, T^{s}_{sub}, P^{s}_{sub}, PT^{S})$, with the requirement $(P^{S1} \cup T^{S1}) \cap (P^{S2} \cup T^{S2}) = \varnothing$ for all $s_1, s_2 \in S$ such that $s_1 \neq s_2$;
2. PT *such that* $T_{sub} \rightarrow S$ is a submodule function that assigns a submodule to each $T_{sub} \subseteq T$ with the requirement that the module hierarchy is acyclic;
3. PS is a port-socket function that is responsible for assigning a port-socket relation $PS(t) \subseteq P_{sock}(t) \times P^{SM(t)}_{port}$ to each substitution transition $t \in T_{sub}$. It is needed that $ST(t) = PT(p'), C(p) = C(p') \wedge I(p)() = I(p')()\ \forall (p, p') \in PS(t) \wedge$ *for all* $t \in T_{sub}$;
4. $FS \subseteq 2^{p}$ is a set of nonempty fusion set such that $C(p) = C(p')$ *and* $I(p)() = I(p')() \forall (p, p') \in fs$ *and* $\forall fs \in FS$.

3.2. Reachability analysis

CPNs can be verified through reachability analysis. Reachability analysis a technique that determines all the states that can be reached from an initial state. This can be used to check for deadlocks and other types of unwanted behavior in the system. Reachability analysis allows calculating all reachable states and all possible system behaviors. CPN offers verification of several properties, such as liveness property, deadlock-freeness property, reachability analysis property, etc.

In safety-critical systems, the system model must be free of deadlocks, and for any reachable state, there should be at least one state for each transition. Therefore, in the behavioral verification of a CPN model, the following properties are analyzed:

Reachability: A certain marking M is called reachable if there exists a firing sequence that leads from the initial marking M_0 to M. Formally, $M \in R(N, M_0)$.

Deadlocks: A marking M in the CPN model is dead if no transition is enabled in that marking. A CPN model is deadlock-free if each reachable marking enables a least one transition.

Dead Transitions: A transition is considered to be dead if there is no reachable marking in which that transition is enabled.

Liveness: A transition t is said to be live if for any marking $M \in R(M_0)$, there exists a sequence of transitions enable from M, which contains t. Formally, $\forall M \in R(N, M_0)$ such that $\exists M \in R(N, M)$: $t' \leq M'$. The liveness property ensures that blocking will never occur in the CPN model.

Boundedness: A place P is called k-bounded for some $k \in N$, if in any reachable marking, place P never has more than k tokens, i.e., $\forall M \in R(N, M_0)$: $M(P) \leq k$. The boundedness property ensures that the number of in-process parts is upper bounded, which in turn ensures the stability of the CPN model (Savi and Xie, 1992).

Dead Markings: The dead marking information tells us about the markings with no enabled transitions.

CPN IDE[1] is a tool suite for modeling, editing, simulation, and state space analysis. Using Cpntool suit, we can generate a state space for a given Petri net model by calculating the reachable states (markings), which makes it possible to answer a large set of verification questions concerning the system's behavior, e.g., deadlocks, liveness, and fairness (Jensen et al., 2007). The state space can also be represented through a directed graph called a state space graph, where nodes of the graph represent states and arcs represent occurring events. State space analysis makes it possible to ensure that all possible executions are covered.

3.3. Human rescue robot system

Autonomous robots have been used in search and rescue operations during disasters such as earthquakes, floods, and hurricanes, as well as in industrial accidents and other emergency situations to increase the speed and effectiveness of search and rescue operations while reducing the risk to human rescuers. For instance, as a rapid response, Miami Dade Fire Rescue (MDRF) primarily carried out tactical drone operations for direct lifesaving and mitigation activities when portions of the surfside's twelve-story Champlain Towers South condominium collapsed on June 24, 2021 (Murphy, 2021).

HRRS is a type of collaborative system that is designed to assist with search and rescue operations in hazardous or inaccessible environments. These systems typically involve a combination of robots, sensor equipment, and human operators working together to locate and rescue victims. The robots used in human rescue robot systems are typically mobile and equipped with cameras, sensors, and other equipment to help them navigate and gather information about the environment (Ali et al., 2022).

This paper uses HRRS as a case study to illustrate our concept of safety for collaborative systems by modeling it in hierarchical CPN. HRRS consists of three types of robots, i.e., Searching Robot (SR), Obstacles Removing Robot (OR), and Lifesaving Robot (LSR) which are controlled by a Control Station (CS) as shown in Fig. 1.

Each robot is considered an independent system collaborating with others to rescue victims from the disaster area. The CS manages overall rescue operations and controls robots. It initiates a rescue operation by sending disaster information, such as location, to the robots. The robots periodically update their status (health diagnostic information, location, and task status) to CS.

The rescuing robots are expected to be able to perform safe operations to rescue victims from the disaster area. All three collaborative systems SR, OR and LSR) interact with each other to save human life. The role and responsibilities of each robot are described below.

SR: The searching robot gets disaster information from CS using CS to Robot (CS2R) communication, searches for victims on the ground, and sends its location to the obstacle-removing robot and lifesaving robot using Robot to Robot (R2R) communication. It also updates its status to CS using Robot to CS (R2CS) communication. SR also updates its status to CS periodically.

OR: It is responsible for removing obstacles from a victim's surroundings so that LSR would quickly rescue the victim without any hurdles. OR receives the victim's location from SR and scans for obstacles surrounding the victim. When it finds obstacles around the victim, it

[1] https://cpnide.org/

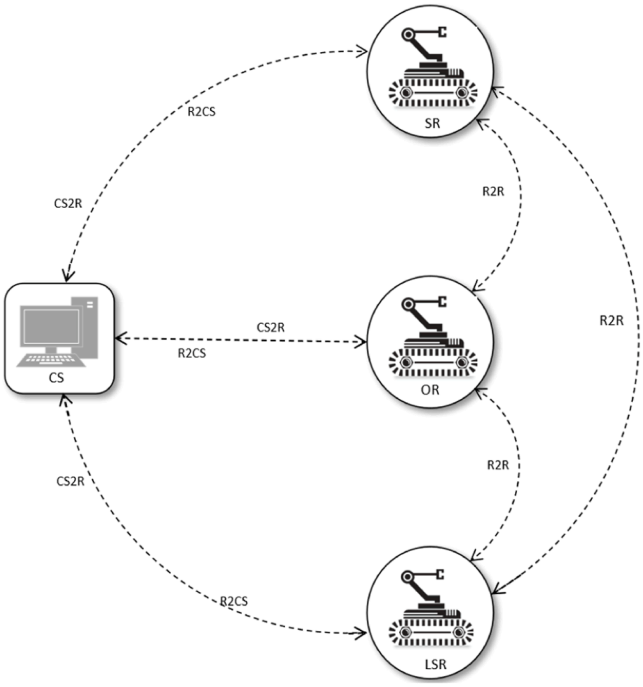


Fig. 1. HRRS high-level architecture.

estimates the shortest path and starts moving towards the victim. After completing its mission, the OR sends a clearance message to the LSR for further operations. The OR also updates its status to CS periodically.

LSR: LSR receives a clearance message and location of the victim from OR, approaches the victim, and evacuates the victim to a designated safe zone. LSR also updates its status to CS periodically.

4. Modeling and safety analysis for HRRS with hierarchical CPN

In this section, we describe the modelling and safety analysis for HRRS. The modelling and safety analysis procedure is mentioned in Fig. 2.

Our modelling and safety analysis procedure has six steps: 1) define the scope of the collaborative system (2) model the collaborative system using hierarchical CPN; 3) model each system that participate in the collaboration; 4) inject faults in each participating system (5) design safety guards for each injected fault and redesign the system; 6) simulate the collaborative system to check the behaviour of the system before and after applying safety guards (validation and verification).

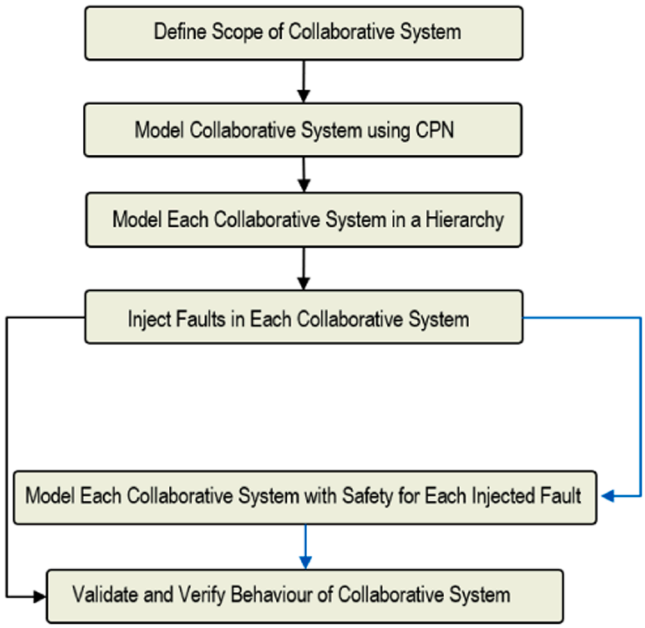


Fig. 2. Detailed procedure of modelling and safety analysis.

The Scope of collaborative system (i.e., HRRS) is described in the Section 3.3. Rest of the steps are covered under this section, Section 5, 6 and 7 of this paper.

Using hierarchal CPN for safety analysis of collaborative systems provides a more detailed view of the system state, which can help identify potential safety hazards more accurately. The additional attributes of tokens, such as colors and data values, can provide a better understanding of the system's behavior and help in identifying potential safety hazards that may be skipped using traditional Petri nets.

Before modeling the system, we make the following assumptions:

- Only one victim is in the scenario at a time.
- Only one type of fault for each robot in one simulation.
- Each robot moves independently from the other, i.e., they do not move together in a group. They just move when the conditions for their function are fulfilled.

During system modeling, we consider the following failure cases at the system level:

- Sensor Failure: Failure due to the problem in sensing the environment that hinders the task in sensing objects of interest.
- Navigation Failure: Failure while navigating to the target location, e.g., slippery ground.
- Object Clearance Failure: Failure in clearing the area due to insufficient actuating mechanism on the robot.

The above assumptions were made for simplification in the initial study, and we plan to relax them in future works and add more failure modes.

4.1. Model structure

In this section, we present a hierarchical CPN model for HRRS. Fig. 3 shows the top-level hierarchical CPN model for collaborative HRRS, which consists of four substitution transitions (rectangles with double-lined borders) representing entities defined for CS, SR, OR, and LSR. The model shows that CS triggers the rescue operation by sending the rescue operation's location. Next, the SR searches for victims and sends the victim position to OR. The OR searches for obstacles around the victim, removes the obstacles, and sends a clearance message along with the victim's position to LSR. Finally, the LSR approaches victims and takes them to a safe area. Each robot reports its status to CS periodically.

4.2. Model description

The model description is divided into 1) color sets, 2) variables, 3) constants, 4) parameters, and 5) functions.

Color Sets: A color set is like a data type that defines the content of a variable. We defined some color sets in the model, as shown in Appendix (Table 3).

Variables: There are some variables defined for modeling the system. The description of variables can be seen in Appendix (Table 4).

Constants: We defined some constants in the model, which are shown in the Appendix (Table 5). These constants are used to represent error conditions and some default values.

Parameters: We defined some parameters in the model, as shown in Appendix (Table 6). These parameters are used as initial values of some places to make it easy to simulate different configurations. If the model needs to be simulated with different configurations, we can simply

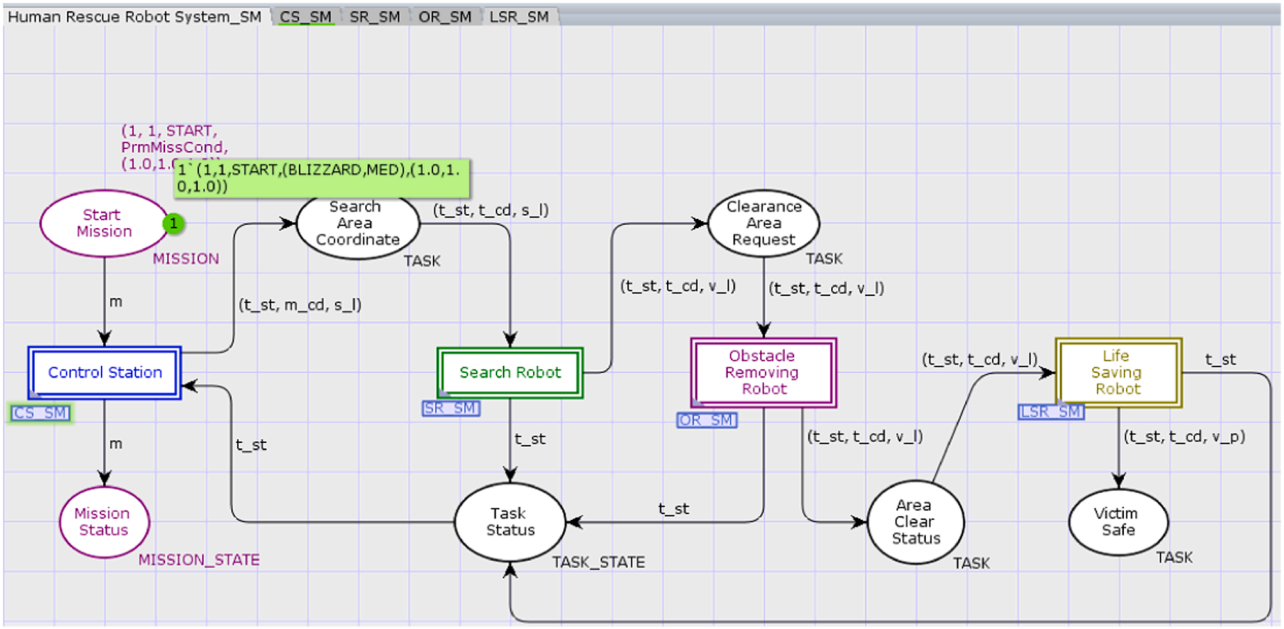


Fig. 3. Hierarchical CPN model for HRRS.

replace the values of the parameters with new values.

Functions: There are some functions defined in the model, as shown in Fig. 4. The model has a complex logic, and these functions are used to make it less cluttered and comprehensible. The description of defined functions are mentioned below.

CalcTaskCondN: The function for calculating a value based on the weather condition and the severity. The value is later used for determining the failure condition with respect to the robot's robustness level in performing a navigation function.

CalcTaskCondS: This function is used for determining the sensor-related failure condition with regard to the robot's robustness level in performing the environment sensing function.

CalcTaskCondO: The function for calculating a value based on the weather condition and the severity. The value is later used for determining the actuating-related failure condition regarding the robot's robustness level in performing object removal functions.

CalcTaskCondV: The function for calculating a value based on the weather condition and the severity. The value is later used to determine the sensor-related failure condition regarding the robot's robustness level in the victim localization function.

HdlErrSrSens: The function for determining the error condition for the SR sensing function.

```
▼Function Declarations
 ▼fun CalcTaskCondN(w, s) =
  (WEATHER.ord(w) * 7) + (SEVERITY.ord(s) * 15);
 ▼fun CalcTaskCondS(w,s) =
  (WEATHER.ord(w) * 8) + (SEVERITY.ord(s) * 17);
 ▼fun CalcTaskCondO(w,s) =
  (WEATHER.ord(w) * 9) + (SEVERITY.ord(s) * 20);
 ▼fun CalcTaskCondV(w,s) =
  (WEATHER.ord(w) * 9) + (SEVERITY.ord(s) * 18);
 ▼fun HdlErrSrSens(x) =
  if x = SYS_ERR_SR_SENS_FAIL
  then FAILED else SUCCESSFUL;
 ▼fun HdlErrSrNav(x) =
  if x = SYS_ERR_SR_NAV_FAIL
  then FAILED else SUCCESSFUL;
 ▼fun HdlErrOrNav(x) =
  if x = SYS_ERR_OR_NAV_FAIL
  then FAILED else SUCCESSFUL;
 ▼fun HdlErrOrObjAct(x) =
  if x = SYS_ERR_OR_OBJ_ACT_FAIL
  then FAILED else SUCCESSFUL;
 ▼fun HdlErrLsrNavRoute(x) =
  if x = SYS_ERR_LSR_NAV_ROUTE_FAIL
  then FAILED else SUCCESSFUL;
 ▼fun HdlErrLsrNavSens(x) =
  if x = SYS_ERR_LSR_NAV_SENS_FAIL
  then FAILED else SUCCESSFUL;
```

```
 ▼fun HdlErrLsrLocVic(x) =
  if x = SYS_ERR_LSR_LOC_VIC_FAIL
  then FAILED else SUCCESSFUL;
 ▼fun HdlNavF(st, w, s, err, r) =
  if CalcTaskCondN(w, s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_SR_NAV_FAIL
  then FAILED
  else SUCCESSFUL;
 ▼fun HdlScanF(st, w, s, err, r) =
  if CalcTaskCondS(w,s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_SR_SENS_FAIL
  then FAILED
  else SUCCESSFUL;
 ▼fun HdlObjF(st, w, s, err, r) =
  if CalcTaskCondO(w,s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_OR_OBJ_ACT_FAIL
  then FAILED
  else SUCCESSFUL;
 ▼fun HdlVicLocF(st, w, s, err, r) =
  if CalcTaskCondV(w,s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_LSR_LOC_VIC_FAIL
  then FAILED
  else SUCCESSFUL;
```

Fig. 4. List of functions defined in HRRS CPN model.

HdlErrSrNav: The function for determining the error condition for the SR navigation function.

HdlErrOrNav: The function for determining the error condition for the OR navigation function.

HdlErrOrObjAct: The function for determining the error condition for OR object-removal function.

HdlErrLsrNavRoute: The function for determining the error condition for the LSR navigation route function.

HdlErrLSrNavSens: The function for determining the error condition for the LSR navigation sensing function.

HdlErrLsrLocVic: The function for determining the error condition for the LSR victim localization function.

HdlNavF: The function for determining the failure condition for the navigation function. This function calculates the failure condition based on the input task state, weather condition, severity, error condition, and the robot robustness level.

HdlScanF: This is the function for determining the failure condition for environmental scanning. This function calculates the failure condition based on the input task state, weather condition, severity, error condition, and the robot robustness level.

HdlObjF: This function is for determining the failure condition for the object-removal. This function calculates the failure condition based on the input task state, weather condition, severity, error condition, and the robot robustness level.

HdlVicLocF: This function determines the failure condition for the victim localization. This function calculates the failure condition based on the input task state, weather condition, severity, error condition, and the robot robustness level.

Table 1 categorizes the functions used in building and simulating HRRS system model.

4.3. Modelling HRRS in hierarchical CPN with failures

As explained below, we incorporated three types of system-level failures into the CPN model.

Sensor Failure: Failures due to the problem in sensing the environment that hinders the task of sensing objects of interest.

Navigation Failure: Failures while navigating to the targeted location, e.g., slippery ground.

Object Clearance Failure: Failure in clearing the area due to

Table 1
Functions and their categorization.

Function Type	Functions	SR	OR	LSR
General	CalcTaskCondN	✓	✓	✓
	CalcTaskCondS	✓	✓	✓
	CalcTaskCondO	✓	✓	✓
	CalcTaskCondV	✓		✓
	HdlNavF	✓	✓	✓
	HdlScanF	✓	✓	✓
	HdlObjF		✓	✓
	HdlVicLocF	✓	✓	✓
Robot Specific	HdlErrSrSens	✓		
	HdlErrSrNav	✓		
	HdlErrOrNav		✓	
	HdlErrOrObjAct		✓	
	HdlErrLsrNavRoute			✓
	HdlErrLsrNavSens			✓
	HdlErrLsrLocVic			✓

insufficient actuating mechanism on the robot.

4.3.1. Hierarchical CPN model for CS

A hierarchical CPN model for CS is shown in Fig. 5. The CS first assesses the search and rescue area, for that it gets mission id (m_id), mission goal (m_g), mission state (m_st), mission conditions (m_cd) and gets mission area coordinates (m_l). The color tokens over *Start Mission* show the weather conditions and their severity at the beginning of the rescue operation. This step is initiated by the operators in the CS. For example, the operator determines the weather conditions (good, wind, rain, fog, and snow) and the severity of weather as low, medium and high. The severity is an enumeration of these values. In the first assessment it checks whether CS has some existing error (err) e.g., location, mission identification and etc. If turns to be true (err = SYS_ERR_CS_TRUE), the simulation does not proceed further. In the next stage, the model has a guard condition to check the state of the mission state (m_st). If the mission state turns to be false and mission state is updated as FAILED (m_st = FAILED), the mission is aborted, and mission state is updated. Otherwise, the CS will start the rescue mission by sending a search area location (s_l) and other information (m_id, m_g, m_st, m_cd) to SR for further processing. The CS also controls overall rescue operations and gets task status (t_st) to monitor the rescue operation. It also gets mission updates from each robot periodically.

4.3.2. Hierarchical CPN model for SR

We modeled SR using hierarchical CPN, as shown in Fig. 6. The SR receives necessary information from the CS that includes task state (t_st), task condition (t_cd), and coordinates for the targeted location (s_l) and navigates to the targeted place to search the potential victims. When SR finds victims, it sends the location information (v_l) of victims and other information to the LSR for rescue and goes to the *Ready* state.

As mentioned above, we incorporated two failure cases, i.e., *Handle Env. Sens Failure* and *Handle Navigation Failure* into the CPN model for SR. Moreover, new types of places and transitions, which are faulty states and transitions, are also added to the CPN model. For example, *Handle Env. Sens Failure* is added to see SR's behavior when it fails to sense the environment due to environmental variabilities such as fog, rain, etc. In addition, *Handle Navigation Failure* fault is also added to see SR's behavior in case SR fails to navigate to a certain destination due to environmental conditions. When any type of the above-mentioned faults occur, the SR updates task status (t_st) to the CS and goes to the *Ready* state.

4.3.3. Hierarchical CPN model for OR

We modeled OR using hierarchical CPN, as shown in Fig. 7. The OR gets victim coordinates (v_l) and other information (t_cd, t_st) from SR,

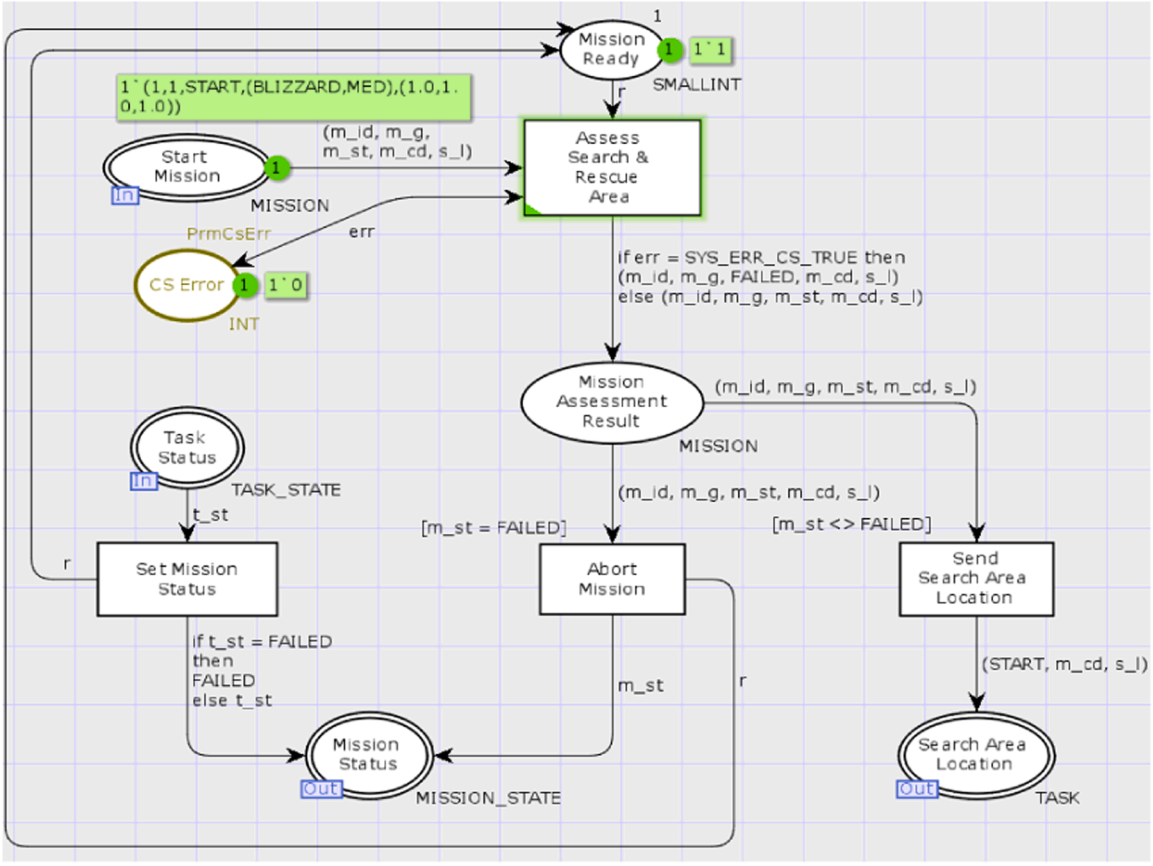


Fig. 5. Hierarchical CPN model for CS.

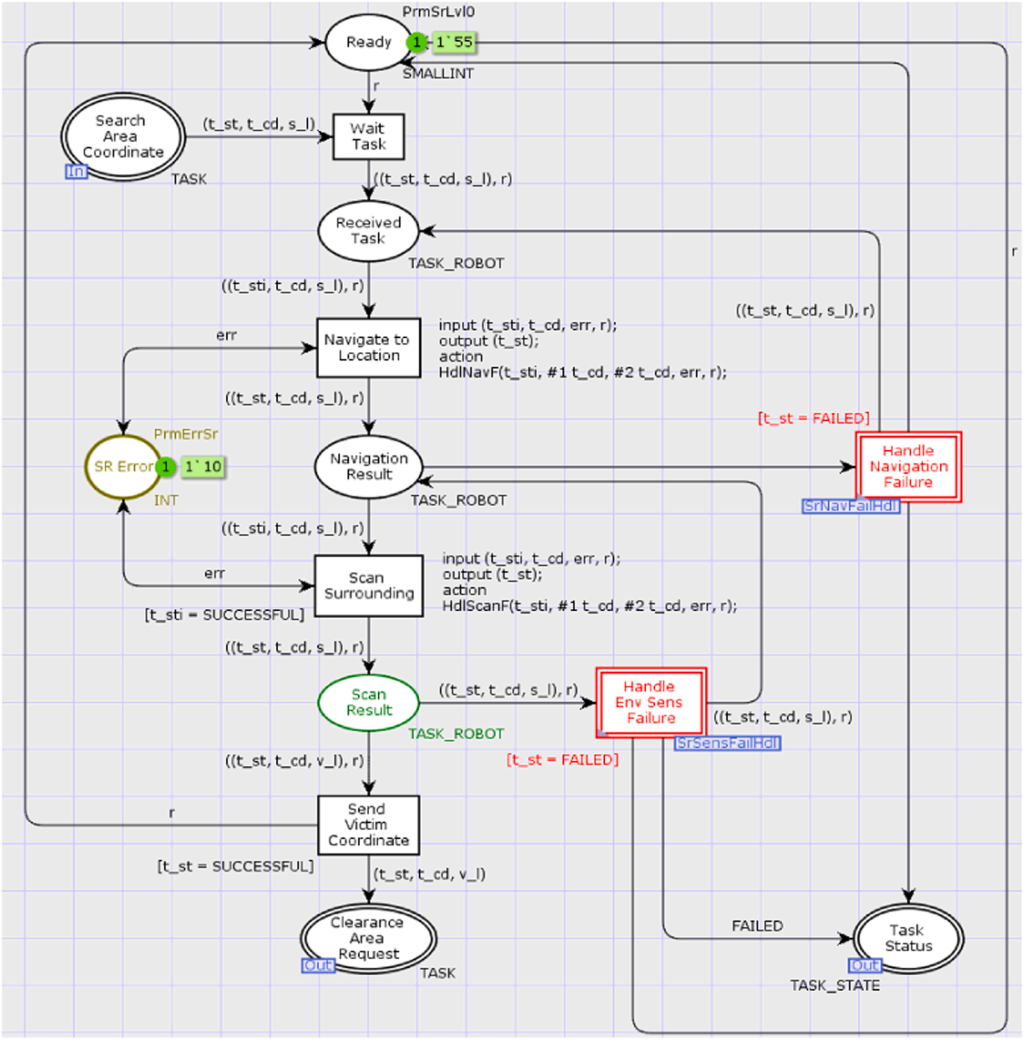


Fig. 6. Hierarchical CPN model for SR.

calculates an optimal route to the victim, searches obstacles around the victim, removes the obstacles, and reports its task status (t_st) to CS. The OR also sends location information of victims (v_l) and other necessary information (t_st, t_cd) to the LSR for further processing.

We inserted two failure cases, i.e., *Handle Obs Clearance problem* and *Handle Navigation Failure,* into the CPN model for OR. If the searching for obstacles mission fails due to a problem in OR, then *Handle Obs Clearance problem* transition can be fired while *Handle Navigation Failure* transition is designed to know the OR's behavior in case of environmental variabilities. In both cases, they update the task status (t_st) back to the CS and go the *Ready* state.

In OR, there is one transition on the top-level model where failure can be simulated, which is the "Clear Location" transition. The failure is simulated through "HdlObjF" as shown in Fig. 12. The "HdlObjF" takes WEATHER (w), SEVERITY (s), the Robot Robustness (r), the TASK_STATE (st), and the ERROR NO (err) parameters to simulate the failure state. CalcTaskCondV function is used to calculate the combination of WEATHER and SEVERITY in order to determine the robustness level of OR at failure state.

4.3.4. Hierarchical CPN model for LSR

The task of LSR is to get the clearance message from OR and to move victims to a safe zone from the disaster zone. Fig. 8 shows the CPN model for LSR. It tells us the operational behavior of the LSR with and without hazardous events. *Handle Env. Sens. Error* and *Handle Nav. Error* are two faults added into the model to see the system's behavior in case of these two faults. In SLR, there is one transition on the top-level model where failure can be simulated, which is the "*Locate Victim*" transition, as shown in Fig. 8.

The sensing failure in LSR is simulated through the function "HdlVicLocF" (see Fig. 4). The "HdlVicLocF" uses some input parameters to simulate a failure state. In the current model, it uses the WEATHER (*w*), SEVERITY (*s*), the Robot Robustness (*r*), the TASK_STATE (*st*), and the ERROR NO (*err*) parameters. If the robot's current configuration, which is characterized by the robot's robustness, is lower than the combination of the current WEATHER and SEVERITY, then LSR will encounter a failure. The combination of the WEATHER and SEVERITY is separately calculated by the function CalcTaskCondV, as shown in Fig. 4. Function CalcTaskCondV is used to calculate the combination of WEATHER and SEVERITY for the victim localization failure.

In SLR, "*Navigation to Victim Location*" transition is where failure can be simulated, as shown in Fig. 8. The navigation failure is simulated through the function "HdlNavF" (see Fig. 4). The "HdlNavF" function takes some input parameters to simulate a failure state. The parameters are WEATHER (*w*), SEVERITY (*s*), robot's robustness (*r*), the TASK_STATE (*st*), and the ERROR (*err*) parameters. If the robot's current configuration, which is characterized by the robot's robustness, is lower than the combination of the current WEATHER and SEVERITY, then the LSR will experience navigation failure. The combination of the WEATHER and SEVERITY is individually calculated by the function

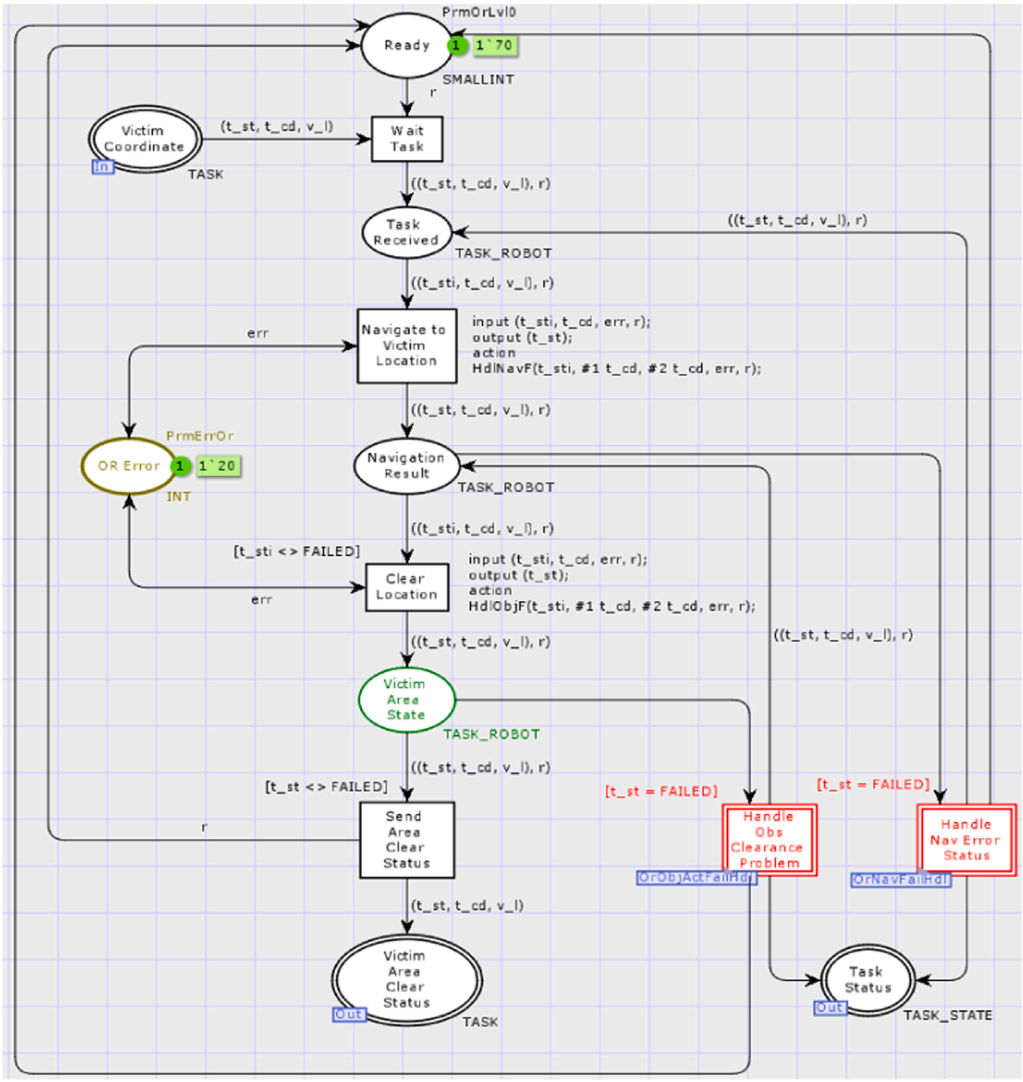


Fig. 7. Hierarchical CPN model for OR.

CalcTaskCondV, as shown in Fig. 4. Function CalcTaskCondV is used to calculate the combination of WEATHER and SEVERITY for the victim localization function.

In order to cope with the navigation failure, we extended the "*Navigation to victim Location*" transition in LSR to another subpage model called Nav2Dest_SM, as shown in Fig. 9. The extended model takes victim's position (v_pos) and task state (t_st) to find the optimal route and drive to the destination otherwise, it reports the navigation problem to the CS.

5. Simulation configurations

As mentioned earlier, the model can be parameterized to allow different configurations for simulating different scenarios. The configurations can be done by changing the parameters of some initial values or by changing the functions to give different results that will change the simulation flow.

Changing Parameters: This modeling type gives the designers flexibility because one does not need to change the initial values of parameters directly in the model. Instead, one can simply change the parameter value and simulate the model for desired values, as shown in Fig. 10.

Changing Functions: Some functions might need to be modified to control the flow of the simulation. The following functions (Fig. 11) can be modified according to our goal.

The results of the above functions are compared to the robot's robustness level, as shown in Fig. 12.

A failure condition is met when the result of a function is over the robot's robustness level. Thus, depending on the set of robot's robustness level, we might need to change the above functions, e.g., to simulate a failure condition for a certain robustness level. The constants used in those functions can be set to any value that meets our simulation goals.

6. Failure handling modelling

In failure handling modeling, we incorporate additional places and transitions to simulate failure handling to enable the simulation to proceed without failure. The following sections explain hierarchical CPN modeling for each collaborative system, i.e., SR, OR and LSR.

6.1. Hierarchical CPN model for SR with safety

When SR faces a failure in "*Scan Surrounding*" (Fig. 6), a safety mechanism called "*Handle Env Sens Failure''* is defined to address sensing failure. In this section, we further extended the "*Handle Env Sens Failure''* transition to ensure safety (Fig. 13). The "*Handle Env Sens Failure*'' transition processes the incoming tokens, i.e., task state (t_st), task

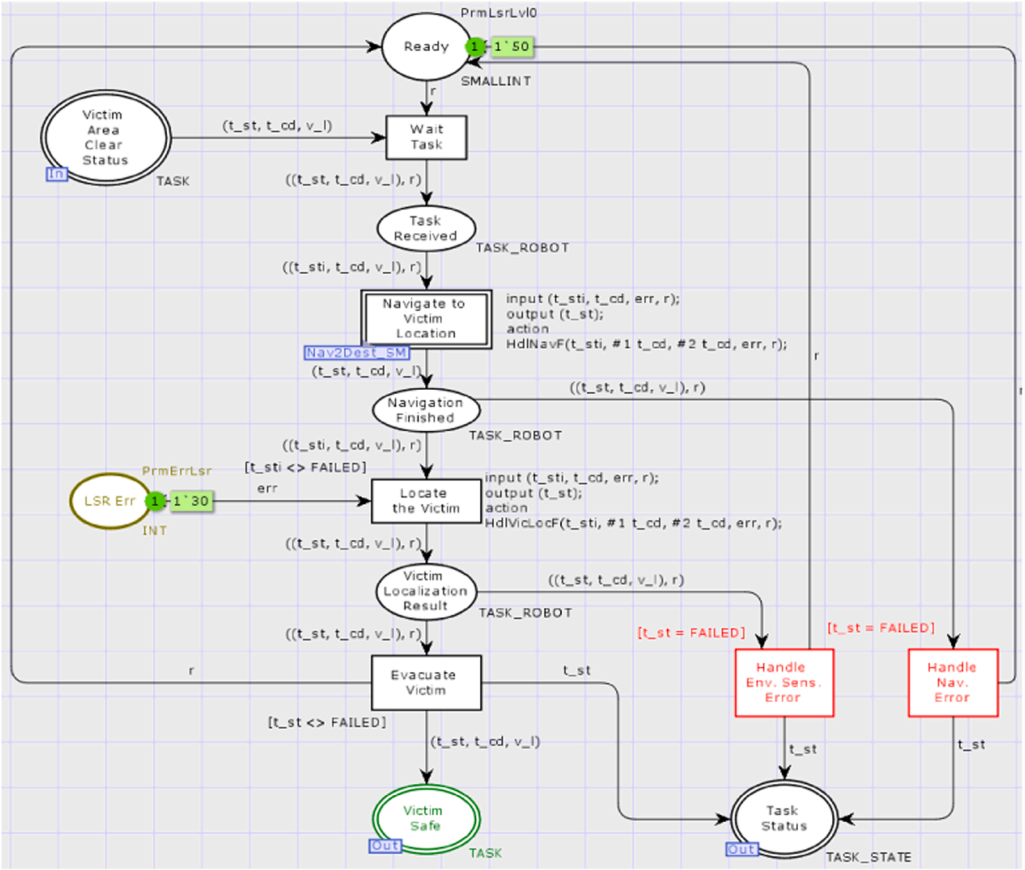


Fig. 8. Hierarchical CPN model for LSR.

condition (t_cd), search location information (s_l), and robot robustness level (r), only in the case of failure when TASK_ROBOT (*t_st*) = FAILED becomes true. In the case of sensing failure, the SR retries three times ($k = 3$) and tries to increase the robot robustness level (r) in each iteration. In the first iteration, the SR tries with normal robot robustness level (r), and if it fails, then in the next iteration, the safety mechanism will increase the robot robustness level (r) to a higher value (PrmLsrLvl1) and try again. This corresponds to, for example, activating additional sensors to sense the environment better in bad weather conditions. The TASK_STATE (*t_st*) needs to be changed to a SUCCESSFUL first to enable the next iteration. In the next iteration, the "Scan Surrounding" transition will again compute the failure state as explained. If the increase of robot robustness level cannot cope with sensing failure, the "*Handle Env Sens Failure*'' transition will be again activated. And inside this transition, the flow will go through the "*Report Scan Failure*" as the condition $k = 3$ is fulfilled, and there is a token in the "Retry Counter" place. In the case of unresolved failure, the TASK_STATE will be set to FAILED, and the robot's robustness level will be reset back to its original value (PrmLsrLvl0).

When the safety mechanism sufficiently addresses the sensing failure, the simulation will continue to the "*Send Victim Coordinate*" transition in the SR model (Fig. 6).

Similarly, when SR faces failure due to the "Navigation to Location" transition. A safety mechanism called "*Handle Navigation Failure*" is defined to cope with this failure. The safety mechanism for transition "*Handle Navigation Failure*" is mentioned in Fig. 14. The "*Handle Navigation Failure*" transition will process the incoming tokens only in the case of failure (TASK_STATE (*t_st*) = FAILED). In the first iteration, the safety mechanism will increase the Robot Robustness level (r) to a higher value (PrmLsrLvl1), which means that the SR will increase traction to be able to navigate better in a bad weather condition. To enable the next iteration, the TASK_STATE (*t_st*) is initially changed to SUCCESSFUL. The "Navigation to Location" transition will again compute the failure state in the next iteration. If the Robot Robustness level rise cannot sufficiently control navigation failure, the "*Handle Navigation Failure*" transition will be triggered again. Inside the "*Handle Navigation Failure*" transition, the simulation flow will go through the "*Report Navigation Failure*" as the condition [$k = 3$] is satisfied, and there is a token in the "Retry Counter" place as shown in Fig. 13. In the case of unresolved failure, the TASK_STATE will be set to FAILED, and the robot's robustness level will be reset to its original value (PrmLsrLvl0). Otherwise, if the safety mechanism is sufficient to cope with the navigation failure, the simulation will continue to the "*Send Victim Coordinate*" transition in the SR model (see Fig. 6).

6.2. Hierarchical CPN model for OR with safety

In the CPN model for OR (Fig. 7), two types of failures, i.e., obstacle clearance failure and navigation failure, can occur in "*Clear Location*" and "*Navigation to Victims Location*" transitions, respectively. The safety mechanism for obstacle clearance failure is shown in Fig. 15.

When obstacle clearance failure occurs in the "*Clear Location*" transition, the "*Handle Obs Clearance Problem*" transition is activated and copes with obstacle clearance failure. The "*Handle Obs Clearance Problem*" transition will process the incoming tokens only in the case of failure (TASK_STATE (*t_st*) = FAILED). In the first iteration, the safety mechanism will increase the robot robustness level *(r)* to a higher value *(PrmLsrLvl1)*. Then, the OR will activate alternate actuators to clear obstacles around the victim. The TASK_STATE (*t_st*) requires to be changed to a SUCCESSFUL first to enable the next iteration. In the next iteration, the "*Handle Obs Clearance Problem*" transition will again compute the failure state, and if the increase of robot robustness level

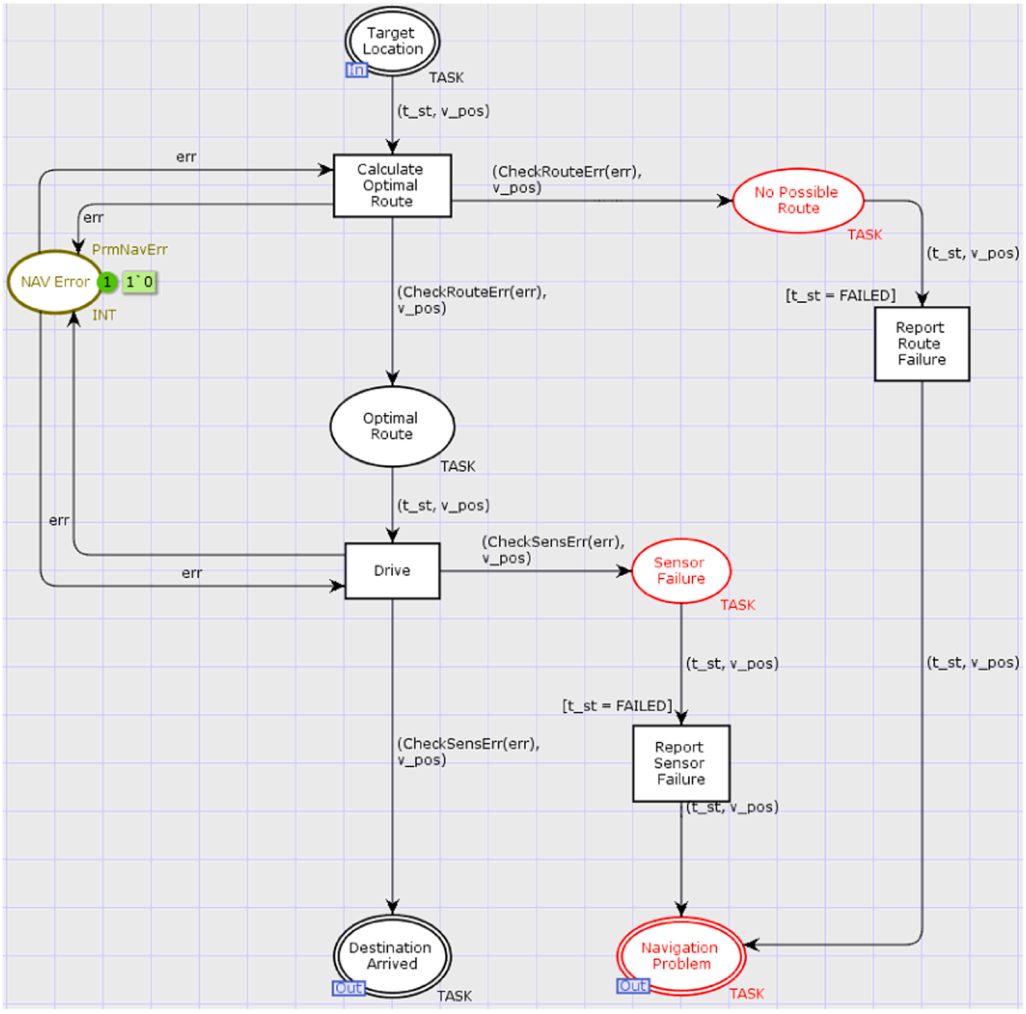
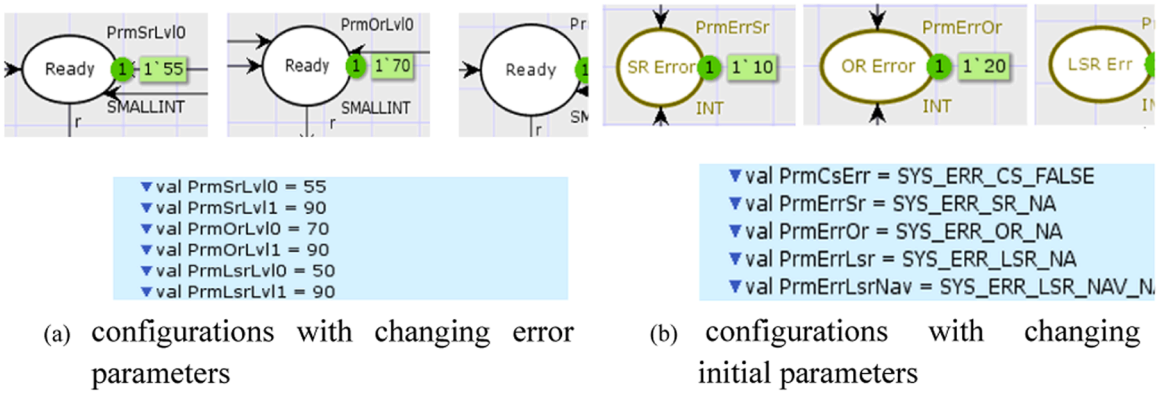


Fig. 9. CPN sub-model for navigation of LSR to a certain destination.



(a) configurations with changing error parameters

(b) configurations with changing initial parameters

Fig. 10. Simulation configurations with changing parameters.

```
fun CalcTaskCondN(w, s) =
  (WEATHER.ord(w) * 7) + (SEVERITY.ord(s) * 15);
fun CalcTaskCondS(w,s) =
  (WEATHER.ord(w) * 8) + (SEVERITY.ord(s) * 17);
fun CalcTaskCondO(w,s) =
  (WEATHER.ord(w) * 9) + (SEVERITY.ord(s) * 20);
fun CalcTaskCondV(w,s) =
  (WEATHER.ord(w) * 9) + (SEVERITY.ord(s) * 18);
```

Fig. 11. Simulation configurations with changing functions.

cannot cope with obstacle clearance failure, the "*Handle Obs Clearance Problem*" transition will be again activated. Inside the "*Handle Obs Clearance Problem*" transition, the flow will go through the "*Report Object clearance Failure*" as the condition [$k = 3$] is fulfilled, and there is a token in the "Fail Counter" place. In the case of unresolved failure, the TASK_STATE will be set to FAILED, and the robot's robustness level will be reset to its original value (PrmLsrLvl0). Otherwise, the "*Send Area Clear Status*" transition will be activated.

Similarly, the "*Navigation to Victims Location*" failure is handled in the

```
fun HdlNavF(st, w, s, err, r) =
  if CalcTaskCondN(w, s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_SR_NAV_FAIL
  then FAILED
  else SUCCESSFUL;
fun HdlScanF(st, w, s, err, r) =
  if CalcTaskCondS(w,s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_SR_SENS_FAIL
  then FAILED
  else SUCCESSFUL;
fun HdlObjF(st, w, s, err, r) =
  if CalcTaskCondO(w,s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_OR_OBJ_ACT_FAIL
  then FAILED
  else SUCCESSFUL;
fun HdlVicLocF(st, w, s, err, r) =
  if CalcTaskCondV(w,s) > r orelse
  st = FAILED orelse
  err = SYS_ERR_OR_OBJ_ACT_FAIL
  then FAILED
  else SUCCESSFUL;
```

Fig. 12. Simulation configuration with failure conditions.

same way as the navigation failure was handled in SR (Fig. 14).

6.3. Hierarchical CPN model for LSR with safety

In the LSR model with a safety mechanism, additional places and transitions were incorporated to simulate safety mechanisms to enable the simulation to proceed without failure. As described earlier (see Fig. 8), when the failure occurs in the "*Navigation to Victim Location*" transition, the "*Handle Nav. Error*" transition is activated to cope with navigation failures. We reused the safety mechanism defined for "*Handle navigation Failure*" in SR (Fig. 14) to cope with the "*Handle Nav. Error*" in LSR. The "*Handle Nav. Error*" transition will process the incoming tokens only in the case of failure *(TASK_STATE (t_st) = FAILED)*. In the first iteration, the safety mechanism will increase the robot robustness level *(r)* to a higher value *(PrmLsrLvl1)*, which means that the LSR will activate more sensors to navigate better in bad weather conditions. Therefore, the *TASK_STATE (t_st)* requires to be changed to a SUCCESSFUL first to enable the next iteration. The "Navigation to Victim Location" transition will again compute the failure state in the next iteration. If the rise of robot's robustness level cannot cope with navigation failure, the "*Handle Nav. Error*" transition will be activated again. Inside the "*Handle Nav. Error*" transition, the flow will go through the "*Report Navigation Failure*" as the condition [$k = 3$] is fulfilled, and there is a token in the "*Retry Counter*" place. In the case of unresolved failure, the *TASK_STATE* will be set to *FAILED*, and the robot's robustness level will be reset back to its original value *(PrmLsrLvl0)*. Otherwise, the "*Evacuate Victim*" transition in the LSR model will be activated (see Fig. 8).

As explained in section 5.3.4, environmental factors may affect the robot's perception of locating the victims. Therefore, we must model the system in a way that works in extreme weather conditions. In SR, we defined "*Handle Env. Sens. Failure*" as a safety mechanism for such kinds of failures (failures related to the environment sensing) to cope with failures related to sensing the environment (Fig. 13). In LSR, we reused the safety mechanism defined for "*Handle Env. Sens. Failure*" in SR (Fig. 13) to cope with the sensing-related failures in LSR.

7. HRRS model validation and verification

In this section, we have identified three requirements for system validation. After the HRRS model is validated, the functional safety is verified using state space analysis in Section 7.1.

7.1. Validation of hierarchical CPN model for HRRS

Design validation is an important step that is required to be performed before any further analysis of safety-critical systems. Validation plays a critical role in ensuring that a system is fit for its intended purpose, satisfies user needs, and operates reliably (Jensen et al., 2007). As mentioned above, here we validate three system safety criteria of the HRRS hierarchal CPN model. The model validation process helps us whether the model satisfies specific safety requirements, which verify the functional safety in Section 7.2. The HRRS model has to satisfy the following three requirements as shown in Table 2. VR1 ensures that all states in HRRS are involved in the state space, VR2 talks about the endless loops in the system, and VR3 shows any abnormal system termination.

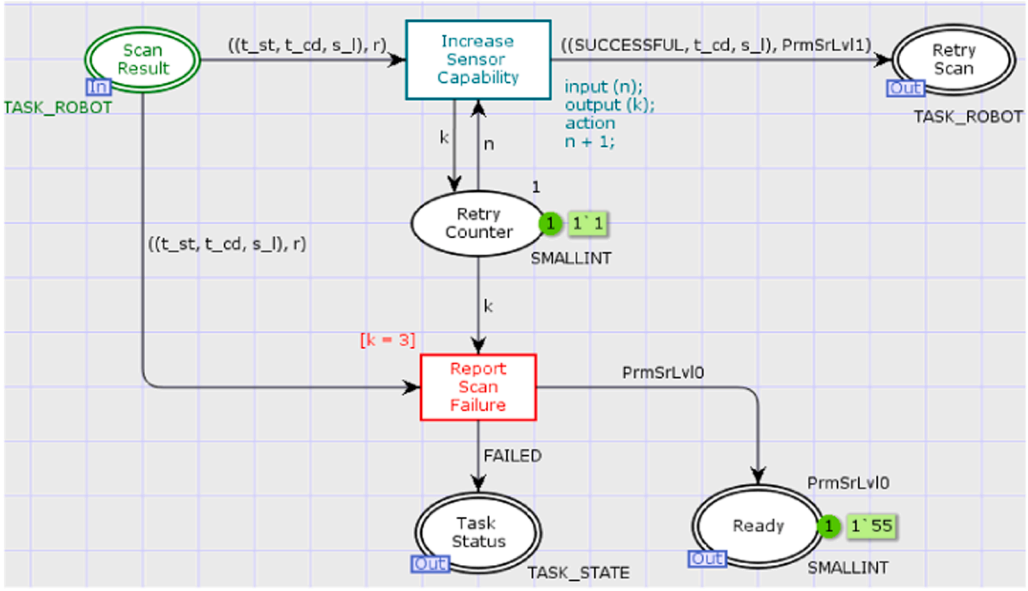


Fig. 13. Safety mechanism to address sensing failure in SR, OR and LSR.

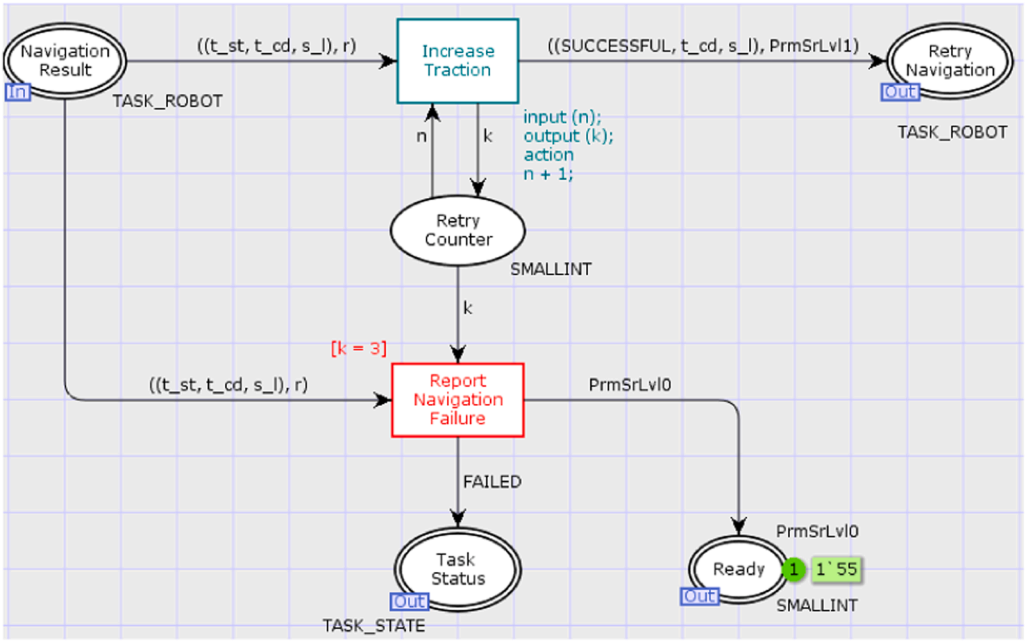


Fig. 14. Safety mechanism for navigation failure in SR, OR and LSR.

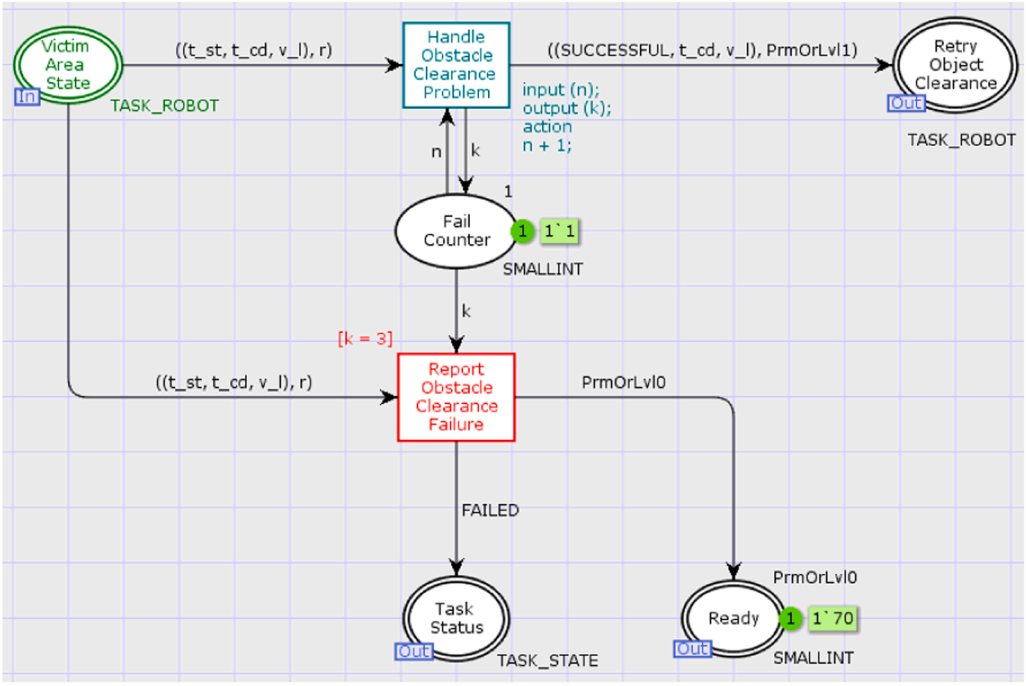


Fig. 15. Safety mechanism for obstacle clearance failure in OR, SR and LSR.

We apply the above three validation requirements to HRRS after designing safety mechanisms for injected failures. The result of VR1 and VR2 for HRRS is presented in Fig. 16 which is generated through state space analysis. For this statistic, we had 199,793 nodes and 200,516 arcs in state space for HRRS. It means that the whole CPN model for HRRS can generate 199,793 different markings, and 200,516 arcs connect the markings. Therefore, the state-space status is *Full*, which tells us that all available states have been calculated in the space.

The Scc (strongly connected component) graph has the same markings and arcs as the state space, meaning there are no cycles in the state space. The Scc indicates that each vertex can be reached from any other vertex, thereby representing that every state or marking within the model is accessible from any other state or marking. The Scc graph has the same markings and arcs as the state-space, which shows there are no cycles or self-loops in the model.

State space is used to verify a formal model with respect to a set of correctness criteria that includes an absence of self-loop and an absence of livelock (Katsaros, 2009; Jensen and Kristensen, 2009a). There are two ways to check the absence of livelocks: If the state space and its Scc graph are isomorphic without any self-loops, then the system model is free from livelocks. However, if the state space contains self-loops or if there's at least one strongly connected component consisting of more

Table 2
Validation requirements and description.

Requirements	Description
VR1: A full state space needs to be calculated	In state space analysis, a full state space needs to be computed which in turn tells whether all variable states are involved.
VR2: Self-loop is discouraged	A self-loop refers to a scenario in which a system retains the same marking while executing a transition, resulting in the utilization of extra system resources.
VR3: absence of livelocks	An important model correctness criterion is the absence of livelocks in the HRRS model

```
Statistics
------------------------------------------------------------------------

State Space
  Nodes:  199793
  Arcs:  200516
  Secs:  300
  Status: Full

Scc Graph
  Nodes:  199793
  Arcs:  200516
  Secs:  16
```

Fig. 16. State space report of HRRS CPN model.

than one node (meaning the number of nodes in the Scc graph is fewer than those in the state space), we must then examine whether all terminal components are trivial.

Livelock in CPN models can occur due to various reasons, such as conflicts in the token colors, incorrect transitions, or issues in the modeling of the system. A livelock is identified when the state space exhibits a cyclic pattern with no markings occurring outside this cycle. In such a scenario, once the cycle is entered, it perpetually repeats without any progress. VR3 can be validated by following queries as shown in Fig. 17. From the results, can see that there is no livelock in the HRRS model. It validates the models correctness in terms of no conflicts in the token colors, incorrect transitions, or issues in the modeling of the system.

7.2. Verification of hierarchical CPN model for HRRS

The model verification focuses on ensuring that the formal representation of the system is accurate and adheres to specific safety property requirements.

Reachability analysis can be used for various purposes, such as verifying that specific properties hold in all reachable states (e.g., safety properties, liveness properties), checking for deadlock conditions, or detecting potential errors or undesirable behaviors. Reachability analysis is a technique used to determine a system's set of reachable states. In hierarchical CPN, this technique can be applied to each level of the hierarchy separately or to the entire system as a whole. One approach to reachability analysis in hierarchical CPN is to use a compositional method, where the reachability of the entire system is constructed from the reachability of its individual components. Another approach is to use a global method, where the entire system is analyzed as a whole. This can be done by generating a reachability graph for the entire system, which represents all the reachable states. We employed this approach to analyze HRRS.

```
fun ListTerminalScc() = predAllScc(SccTerminal);
fun InValidTerminalScc()= predAllScc(ListTerminalScc(),
   fn n => not (SccTerminal n), NoLimit);
let
val fid = TextIO.openOut "Livelock.txt"
val _ = if InValidTerminalScc()=[]
 then TextIO.output(fid, "Model is Free of Livelock")
 else TextIO.output(fid, "Model has Livelock")
in
TextIO.closeOut(fid)
end
.....
Model is Free of Livelock
```

Fig. 17. Absence of livelock in HRRS CPN model.

Once the reachability graph has been generated, it can be analyzed to check for certain properties, such as liveness, boundedness, or safety properties.

The process of reachability analysis in hierarchical CPNs typically involves the following steps:

1. Construct a CPN model of the system: The system is modeled as a CPN, with places representing the states and transitions representing the events that can change the state.
2. Identify the initial state: The system's initial state is identified and marked in the CPN model.
3. Generate the state space: The system's state space is generated by finding all the reachable states from the initial state.
4. Check for unwanted behaviors: Once the state space is generated, it can be analyzed to check for unwanted behaviors such as deadlocks.

We followed the above steps to take advantage of state-space analysis to verify whether the hazardous transitions are reachable in the HRRS model. The state-space identifies all possible reachable states and transitions the system can reach from the initial state through transition firings. The reachable states and state changes are represented by a directed graph where the nodes are the set of reachable markings, and the arcs are the binding elements. In addition, state-space includes necessary information and properties, including the CPN model's home properties, liveness, fairness, statistics, and boundedness properties. Hence, HRRS hierarchical CPN model is verified against three safety properties requirements:

- Dead Transitions
- Boundedness
- Deadlock

First of all, we generated the state space report for HRRS with injected failure. For this statistic, we had 107 nodes and 106 arcs in the state space. It means that the whole CPN model for HRRS can generate 107 different markings, and 106 arcs connect the markings. Therefore, the state-space status is *Full*, which tells us that all available states have been calculated in the space.

The Scc graph has the same markings and arcs as the state space, meaning there are no cycles in the state space. The liveness properties specify some dead markings and dead transition instances in the generated state space. In this case, we got 100 dead markings and 27 dead marking instances. The dead markings may indicate some deadlocks and the deadlock instances confirm where the potential deadlocks are in the model. Dead marking is a unique marking; it shows that the system is terminated, and no more subsequent actions are available. However, unnecessary dead markings are undesired and can be a design defect. For example, among the dead marking instances, "*Handle_Env*" shows a faulty transition, which is not fired, as a result, LSR could not complete its assigned task.

The dead transitions do not always refer to the deadlocks in the system or the presence of a fault in the system; some time it tells us that a faulty transition is not reachable in the model. For instance,

Abort_Mission is a deadlock transition instance. This is a faulty transition; rightly, it should not be reachable in the model (also refer to Fig. 5). However, dead transitions may indicate the presence of a fault in the system. For example, the *"Evacuate_Victim"* transition is a dead transition, which means that LSR terminated its mission somewhere in the middle of the task due to this failure. Therefore, the model without a safety mechanism needs a through investigation into the dead markings and dead marking instances to rectify the potential design defects.

We also generated the state space graph for HRRS with injected failures as shown in Fig. 18. From the result, it is clear that after the transition *SRNavigate_to_Location*, the SR could not get the location of victims due to environment sensing failure, which can be seen from the *SRHandle_Env_Sens_Failure* transition. We also see that the robot increases its robustness level ($r = 55$) but could not detect the victims around the location due to environmental factors. As a result, the mission was failed as we see from the *CSSet_Mission_Status* transition where task state (t_st=FAILED) is updated as failed.

As anticipated, the system has dead markings and dead transitions, which is not desirable in the model. Therefore, it proves the designed model is incorrect, and the designed model must be revised to ensure that the faulty transitions are not fired.

In order to remove the design defect from the model, we revise the HRRS CPN model. In our CPN model, we defined safety guards for each type of failure and incorporated them into the CPN model to ensure safety. *"Handle Env Sens Failure"* and *"Handle Navigation Failure"* transitions are expanded to define safety guards for sensing failure and navigation failure in SR (see Figs. 13, 14). Similarly, the "*Handle Obs Clearance Problem"* transition is further expanded, and OR's robustness level was increased to activate more actuators to handle the obstacle clearance problem in OR (Fig. 15). Furthermore, *"Handle Env.Sens. Error"* transition was expanded in the CPN model for LSR (Fig. 8), and a sub-model was designed to address the environmental sensing problem in LSR (Fig. 13). After redesigning the CPN model with safety, a state-space report was again generated to see whether faulty transitions were dead and other deadlocks were removed. Fig. 19 shows information on the state-space report for the modified hierarchical CPN model for HRRS. The generated state-space has 199,793 markings and 200,516 arcs. The construction of state-space took 300 s (Fig. 19). This is due to the fact that the modified hierarchical CPN model for HRRS was complex.

Dead Transitions: Detecting and handling dead transitions in a CPN model is important to ensure that the modelled system operates correctly and does not become stuck in an undesirable state. As we mentioned, without safety mechanisms for injected failures, we got 27 dead markings that led to the system failure. After designing the safety mechanisms for each injected failure, we simulated the system and generated state space report as shown in Fig. 19. We see that there are seven dead transition instances, no home markings, and no live transition instances. The fairness properties show that there is no infinite occurrence if sequences as well.

In some cases, the presence of dead transitions might indicate a design issue in the CPN model. We reviewed the overall system design to ensure it accurately reflects the intended behavior and requirements. Therefore, we investigated the dead transition instances i.e. *CS_SM'Abort_Mission, LsrNavFailHdl'Increase_Traction, LsrNavFailHdl'Report_Navigation_Failure, Nav2Dest_SM'*

Report_Route_Failure, Nav2Dest_SM'Report_Sensor_Failure, SrNavFailHdl'Increase_Traction, and *SrNavFailHdl'Report_Navigation_Failure*. During our investigation, we have seen that these dead transition instances show that the faulty transitions must be dead, meaning that these faulty transitions should not be enabled in the presence of defined safety guards. For instance, the *"Report_Sensor_Failure", "Handle_Obs_Clearance_Problem,"* and "*Report_Obstacle_Clearance_Failure"* transitions are dead along with other faulty states, which means that faults in HRRS are acceptably mitigated by defining safety mechanisms.

Boundedness can help in ensuring the correct behavior of a system at a time point. The state transition for HRRS is shown in Fig. 3. We see that the initial state (Start_Mission) is 1`(1,1,START,(BLIZZARD,MED, 1.0,1.0,1.0)). The mission starts when it gets weather conditions, severity, location, and other parameter for the initial condition of the mission condition. It is obvious that there must be always one and only one multiset token on *Start_Mission* place because the mission only starts when all parameters are available. Here, we can apply the boundedness property to verify this function. The boundness properties can specify how many and which tokens a place holds (Jensen and Kristensen, 2009b), it has the capacity to manifest either hardware or software resource demands, as a higher token count in one location implies a greater need for storage space. We run a query to verify the boundness on *Start_Mission* place as shown in Fig. 20. As shown in the results, both the upper bound and lower multiset are the same on the same place *Mark.Human_Rescue_Robot_System'*

Start_Mission. It verifies that the model's operation aligns with the anticipated behavior.

Deadlock: The absence of deadlock markings in a CPN means that the model has been designed or configured in such a way that deadlock situations do not occur. This is essential for ensuring the proper functioning and liveness of a system represented by the hierarchal CPN. Achieving the absence of deadlock markings involves careful modeling and analysis to ensure that proper synchronization mechanisms, resource allocation, and transition firing conditions are in place to prevent processes from getting stuck in a deadlock state. It ensures that the CPN can always make progress and reach its desired states without being indefinitely blocked. Algorithms such as (Katsaros, 2009) are used to identify the deadlocks in the HRRS hierarchal CPN model. Fig. 21 shows the non-standard space query verifying that the HRRS hierarchal CPN model does not include deadlock markings. We verified that all dead markings are correct, meaning that the communication channels have valid participant's state combinations. We see that, when the transition *Life Saving Robot* ends with the status *successful*, the *Mission Status* also produces the same status because *TASK_STATE* (*t_st*) is updated back to the CS. When the *Mission Status* goes to the *FAILED* state it means that a failure has been encountered in the system or some deadlock is occurred (Fig. 18). However, from the result, we see that the system experienced no failure as we see the markings at the top model (Human_Rescue_Robot_System_SM) produce normal outputs and produced no deadlocks. This is due to the fact that the unnecessary dead markings were removed by revising the HRRS CPN model.

7.3. Discussion

From the verification and validation results, we can see that the designed safety mechanisms have mitigated the injected failures. As we expected, the number of live/dead transition instances and the fairness properties are desirable which verifies the correctness of our revised HRRS CPN model. The verification results against deadlock markings, dead transitions, and boundedness properties show the correctness of the revised HRRS model. Validation plays a critical role in ensuring that a system is fit for its intended purpose and satisfies user needs. We also validated the HRRS CPN model against three defined requirements (VR1, VR1, and VR3) that show its fitness for the intended use.

Although we have carefully designed our model and conducted the simulation to verify and validate the proposed approach, several challenges remain. We point out the high dependency on CPN simulator where the real situation can not be reflected due to modelling limitations in CPN IDE. Additionally, we can not guarantee that the proposed solution will be applied for all collaborative scenarios because it has been only applied to the HRRS case study. However, the safety mechanism for injected failures can be applied to mitigate similar failures in other case studies as well.

Along with the numerous benefits, there is a big disadvantage of CPN analysis i.e., the sate space explosion problem (Valmari, 1996; Clarke et al., 2001) that results in the great computational complexity of the

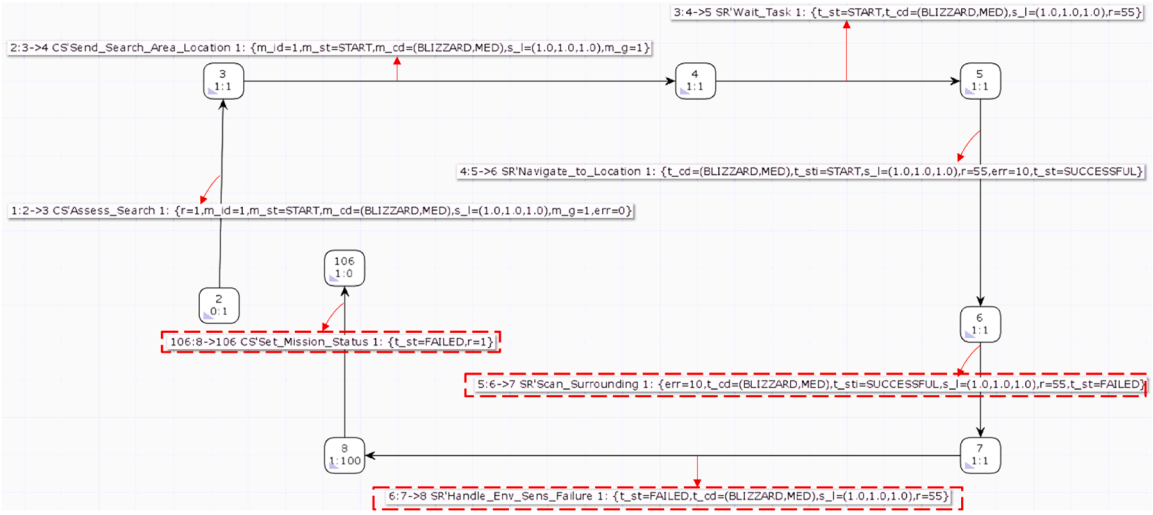


Fig. 18. Details of checking results in state space graph.

```
Statistics
------------------------------------------------------------------------

State Space
  Nodes:  199793
  Arcs:   200516
  Secs:   300
  Status: Partial

Scc Graph
  Nodes:  199793
  Arcs:   200516
  Secs:   16

Home Properties
------------------------------------------------------------------------

Home Markings
  None

Liveness Properties
------------------------------------------------------------------------

Dead Transition Instances
  CS_SM'Abort_Mission 1
  LsrNavFailHdl'Increase_Traction 1
  LsrNavFailHdl'Report_Navigation_Failure 1
  Nav2Dest_SM'Report_Route_Failure 1
  Nav2Dest_SM'Report_Sensor_Failure 1
  SrNavFailHdl'Increase_Traction 1
  SrNavFailHdl'Report_Navigation_Failure 1

Live Transition Instances
  None

Fairness Properties
------------------------------------------------------------------------

  No infinite occurrence sequences.
```

Fig. 19. State space report after mitigating the injected failures.

```
let
val fid = TextIO.openOut "verifcation.txt"
val _ = INT.output(fid,("System´s Boundness: \n")
val _ = INT.output(fid,("Upper Multi-set Bound: \n")
val _ = INT.output(fid,UpperMultisetBound (Mark.Human_Rescue_Robot_System_SM'Start_Mission 1))
val _ = INT.output(fid,("Lower Multi-set Bound: \n")
val _ = INT.output(fid,UpperMultisetBound (Mark.Human_Rescue_Robot_System_SM'Start_Mission 1))
in
TextIO.closeOut(fid)
end
.....
System´s Boundness:
Upper Multi-set Bound:   1`(1,1,START,(BLIZZARD,MED),(1.0,1.0,1.0))
Lower Multi-set Bound:   1`(1,1,START,(BLIZZARD,MED),(1.0,1.0,1.0))
```

Fig. 20. Boundedness properties of a place.

analysis. Especially, this problem occurs in Petri nets with timed properties (Sloan and Buy, 1996; Luo et al., 2014). To reduce this risk, we used a hierarchal approach to better manage the system and to avoid unnecessary states. Hierarchal modelling in CPN can manage and reduce the state space of a complex system by breaking it down into smaller, more manageable sub-models. This hierarchical structuring can help control state space explosion, making it easier to analyse and verify the system. We also used multiset bounds to reduce the state space graph by bounding the number of tokens in certain places. By limiting the possible multiset sizes in specific places, we can reduce the combinatorial explosion of states in our CPN model. This is particularly useful when we have places with large or unbounded token populations that can lead to

```
fun InValidTermina n=  (length (hd (Mark.  Human_Rescue_Robot_System_SM'Search_Area_Coordinates 1 n))= [(t_st, m_cd, s_l)] andalso
                        (hd (Mark.  Human_Rescue_Robot_System_SM'Clearance_Area_Request 1 n))= [(t_st, t_cd, v_l)] andalso
                        (hd (Mark.  Human_Rescue_Robot_System_SM'Area_Clear_Status 1 n))= [(t_st, t_cd, v_l)] andalso
                       (length (hd (Mark.  Human_Rescue_Robot_System_SM'Victim_Safe1 n))= [(t_st, t_cd, v_p)] andalso
                       (length (hd (Mark.  Human_Rescue_Robot_System_SM'Task_Status1 n))= [t_st] andalso
                       (length (hd (Mark.  Human_Rescue_Robot_System_SM'Mission_Status1 n))= [m] andalso
let
val fid = TextIO.openOut "Deadlock.txt"
val _ = TextIO.output(fid, "List of deadlock Markings: \n")
val _ = EvalNodes(InValidTerminal()=ListDeadMarkings(), fn n => not(ValidTerminal n), NoLimit);
        fn n => INT.output (fid n))
in
TextIO.closeOut(fid)
end
.....
List of deadlock Markings:
```

Fig. 21. Absence of deadlock markings.

a vast state space.

8. Conclusions

In this paper, it has been shown how hierarchical CPN can be used to model, validate and verify collaborative safety-critical systems. Different from other approaches, a unique approach has been presented to carry out simulation-based safety verification and validation. We illustrate this approach using HRRS as a case study where we model the HRRS using hierarchical CPN. After modeling, we injected three kinds of failures at the system level and generated the state space to check if there were dead markings, self-loops, or dead transitions. The simulation results indicated that there were no self-loops; however, several dead transitions led to the mission failure of the HRRS. We identified the failure transitions, developed safety mechanisms for each type of failure, and again generated the state space to ensure the safe operation of HRRS to complete its common mission without failures. The simulated results show that all the identified failures were mitigated, and HRRS completed its mission without failure. It was found that the approach based on formal methods (CPN modeling) can be used for the safety analysis, modeling, and verification of collaborative safety-critical systems like HRRS. The hierarchical CPN provides a rigorous way of modeling to implement complex collaborative systems. However, we have seen that CPNs can be computationally expensive to analyze, which can be a limitation for large and complex systems. But it allows for modeling large and complex systems by breaking them down into smaller, more manageable subnets.

In general, hierarchical CPNs can be a useful tool for safety analysis in collaborative systems, but they may not be the best choice for all types of systems or all types of analysis. The choice of CPNs or other formal methods may depend on the specific characteristics of the system to be analyzed and the goals of the analysis.

In the future, we want to include scenarios with timing to make it a timed-CPN. For instance, we want to check different weather conditions after certain times to make it more resilient in adverse weather conditions.

CRediT authorship contribution statement

Nazakat Ali: Conceptualization, Formal analysis, Investigation, Software, Writing – original draft, Writing – review & editing, Methodology. **Sasikumar Punnekkat:** Writing – review & editing, Resources, Supervision. **Abdul Rauf:** Validation, Writing – review & editing.

Declaration of competing interest

We declare no conflict of ineterst.

Data availability

Data will be made available on request.

Acknowledgment

This research was partially supported by the SSF funded DAISY Project.

Appendix

Table 3
Color set and their meaning.

Color Set	Meaning
SMALLINT	A color set for a small integer value, from 1 to 100
LOC	A location in 3D, e.g. (x, y, z) or (latitude, longitude, elevation).
SEARCH_LOC	A color set for the search location. It is based on the LOC color set.
VICTIM	A color set for representing an ID of a victim.
STATE	A color set for representing different possible states. It is an enumeration with possible values: START, RUNNING, PAUSED, SUCCESSFUL, FAILED.
SEVERITY	A color set for representing the different severity levels, for example, the severity of the weather. It is an enumeration of values: LOW, MED, and HIGH
WEATHER	A color set for representing weather conditions. It is an enumeration of values: GOOD, WIND, RAIN, FOG, and SNOW.
BLIZZARD	The weather condition can cause hazardous conditions; thus, the values are ordered based on the possible hazardous condition
WEATHER_STATE	A color set for representing the state of the weather. It is based on the WEATHER and SEVERITY color set.
MISSION_STATE	A color set for representing the state of a mission. It is based on the STATE color set.
TASK_STATE	A color set for representing the state of a task. It is based on the STATE color set.
TASK_COND	A color set for representing the condition faced by a task. It is based on the WEATHER_STATE color set
TASK	A color set for representing a TASK performed by a robot. It is based on three color sets: TASK_STATE, TASK_COND, and SEARCH_LOC.
TASK_ROBOT	An extension of the TASK color set that comprises the TASK and the SMALLINT color set. The SMALLINT component is used to store the Robustness Level of the robot.
VICTIMPOS	A color set for representing the victim's location. It is based on the LOC color set.

(*continued on next page*)

Table 3 (*continued*)

Color Set	Meaning
MISSION_ID	A color set for representing the mission identification number. It is based on the INT color set.
MISSION_GOAL	A color set for representing the mission's goal. It is currently set to the VICTIM color set.
MISSION_COND	A color set for representing the environmental condition faced by a mission. It is based on the WEATHER_STATE color set.
MISSION	A color set for representing a mission. It consists of five color sets: MISSION_ID, MISSION_GOAL, MISSION_STATE, MISSION_COND, and SEARCH_LOC.

Table 4
Variables and their meaning.

Variables	Meaning
m	The variable for storing the MISSION information
v_l	The variable for storing the victim position information
s_l	The variable for storing the search location information
m_id	The variable for storing the mission ID
m_g	The variable for storing the mission goal.
m_st	The variable for storing the mission state
m_cd	The variable for storing the mission condition.
t_st	The variable for storing the task state
t_sti	The variable for storing the task state that will go into a transition and used as an input of a function.
t_cd	The variable for storing the task condition.
err	The variable for storing error value.
r	The variable for storing the robustness level of a robot
ri	The variable for storing the robustness level of a robot that will go into a transition and used as an input of a function.
k	The variable for storing the number of retries due to a failure.
n	The variable for storing the number of retries due to a failure. The value is always k – 1.

Table 5
Constants and their meaning.

Constants	Meaning
SYS_ERR_CS_FALSE	The constant for no system error in CS
SYS_ERR_CS_TRUE	The constant for existing system error in CS
SYS_ERR_SR_NA	The constant for no system error in SR
SYS_ERR_SR_NAV_FAIL	The constant for navigation failure in SR.
SYS_ERR_SR_SENS_FAIL	The constant for sensor failure in SR
SYS_ERR_OR_NA	The constant for no system error in OR
SYS_ERR_OR_NAV_FAIL	The constant for navigation failure in OR
SYS_ERR_OR_SENS_FAIL	The constant for sensor failure in OR
SYS_ERR_OR_OBJ_ACT_FAIL	The constant for a failure with obstacle removing mechanism in OR
SYS_ERR_LSR_NA	The constant for no system error in LSR
SYS_ERR_LSR_EVAC_VIC_FAIL	The constant for a failure with the victim evacuation mechanism in LSR
SYS_ERR_LSR_LOC_VIC_FAIL	The constant for a failure with the victim localization mechanism in LSR
SYS_ERR_LSR_NAV_NA	The constant for no navigation failure in LSR
SYS_ERR_LSR_NAV_ROUTE_FAIL	The constant for a failure with the routing calculation and following in LSR
SYS_ERR_LSR_NAV_SENS_FAIL	The constant for a failure with the sensing for navigation in LSR

Table 6
Parameters and their meaning.

Parameters	Meaning
PrmMissCond	The parameter for the initial condition of the mission condition
PrmCsErr	The parameter of the error condition of the CS
PrmErrSr	The parameter of the error condition of the SR
PrmErrOr	The parameter of the error condition of the OR
PrmErrLsr	The parameter of the error condition of the LSR
PrmErrLsrNav	The parameter of the error condition of the navigation function of the LSR
PrmSrLvl0	The robustness level of SR robot. This parameter is used as the initial value of the SR
PrmSrLvl1	The robustness level of SR. This parameter is used as the update value of the safety mechanism of the SR
PrmOrLvl0:	The robustness level of OR robot. This parameter is used as the initial value of the OR
PrmOrLvl1	The robustness level of OR. This parameter is used as the update value of the safety mechanism of the OR
PrmLsrLvl0	The robustness level of LSR. This parameter is used as the initial value of the LSR
PrmLsrLvl1	The robustness level of LSR. This parameter is used as the update value of the safety mechanism of the LSR

References

Akhtar, N., Rehman, A., Hussnain, M., Rohail, S., Missen, M.S., Nasir, M., Hayder, A., Salamat, N., Pasha, M., 2019. Hierarchical coloured petri-net based multi-agent system for flood monitoring, prediction, and rescue (fmpr). IEEE Access 7, 180544–180557.

Ali, N., Hussain, M., Hong, J.E., 2021. Fault-tolerance by resilient state transition for collaborative cyber-physical systems. Mathematics 9, 2851.

Ali, N., Hussain, M., Hong, J.E., 2022. SafeSoCPS: a composite safety analysis approach for system of cyber-physical systems. Sensors 22, 4474.

An, Y., Wu, N., Zhao, X., Li, X., Chen, P., 2018. Hierarchical colored petri nets for modeling and analysis of transit signal priority control systems. Appl. Sci. 8, 141.

Bowen, J., Stavridou, V., 1993. Safety-critical systems, formal methods and standards. Softw. Eng. J. 8, 189–209.

Clarke, E.; Grumberg, O.; Jha, S.; Lu, Y.; Veith, H. Progress on the state explosion problem in model checking. Informatics: 10 years back, 10 years ahead 2001, 176–194.

Gonçalves, P., Sobral, J., Ferreira, L.A., 2017. Unmanned aerial vehicle safety assessment modelling through Petri nets. Reliab. Eng. Syst. Saf. 167, 383–393.

Gualtieri, L., Rauch, E., Vidoni, R., 2021. Emerging research fields in safety and ergonomics in industrial collaborative robotics: a systematic literature review. Robot. Comput. Integr. Manuf. 67, 101998.

Honour, E. Verification and validation issues in systems of systems. arXiv preprint arXiv: 1311.3626 2013.

Hu, S., Wu, D., Wang, H., 2017. Safety analysis of train control system based on colored petri nets and system-theoretic process analysis. In: Proceedings of the International Conference on Electrical and Information Technologies for Rail Transportation, pp. 175–184.

Hussain, M., Ali, N., Hong, J.E., 2022. Vision beyond the field-of-view: a collaborative perception system to improve safety of intelligent cyber-physical systems. Sensors 22, 6610.

Jensen, K., Kristensen, L.M., 2009a. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Science & Business Media.

Jensen, K., Kristensen, L.M., 2009b. Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer-Verlag, Aarhus, Denmark.

Jensen, K., Kristensen, L.M., 2015. Colored Petri nets: a graphical language for formal modeling and validation of concurrent systems. Commun. ACM 58, 61–70.

Jensen, K., Kristensen, L.M., Wells, L., 2007. Coloured Petri nets and CPN tools for modelling and validation of concurrent systems. Int. J. Softw. Tools Technol. Trans. 9, 213–254.

Katsaros, P., 2009. A roadmap to electronic payment transaction guarantees and a Colored Petri Net model checking approach. Inf. Softw. Technol. 51, 235–257.

Kuo, C.H., Huang, H.P., 2000. Failure modeling and process monitoring for flexible manufacturing systems using colored timed Petri nets. IEEE Trans. Robot. Autom. 16, 301–312.

Lee, J., Bagheri, B., Kao, H.A., 2015. A cyber-physical systems architecture for industry 4.0-based manufacturing systems. Manuf. Lett. 3, 18–23.

Li, Z., Wang, S., Zhao, T., Liu, B., 2016. A hazard analysis via an improved timed colored petri net with time–space coupling safety constraint. Chin. J. Aeronaut. 29, 1027–1041.

Liu, S. Formal modeling and analysis techniques for high level Petri nets. 2014.

Luo, J., Xing, K., Zhou, M., Li, X., Wang, X., 2014. Deadlock-free scheduling of automated manufacturing systems using Petri nets and hybrid heuristic search. IEEE Trans. Syst. Man Cybern. Syst. 45, 530–541.

Maier, M.W., 1998. Architecting principles for systems-of-systems. Syst. Eng. J. Int. Counc. Syst. Eng. 1, 267–284.

Murphy, R., 2021. How robots helped out after the surfside condo collapse. IEEE Spectr. https://spectrum.ieee.org/building-collapse-surfside-robots.

Peterson, J.L., 1977. Petri nets. ACM Comput. Surv. (CSUR) 9, 223–252.

Raman, R., Murugesan, A., 2022. Framework for complex SoS emergent behavior evolution using deep reinforcement learning. In: Proceedings of the INCOSE International Symposium, pp. 809–823.

Savi, V.M., Xie, X., 1992. Liveness and boundedness analysis for Petri nets with event graph modules. In: Proceedings of the International Conference on Application and Theory of Petri Nets, pp. 328–347.

Sloan, R.H., Buy, U., 1996. Reduction rules for time Petri nets. Acta Inform. 33, 687–706.

Song, H., Schnieder, E., 2018. Evaluating fault tree by means of colored Petri nets to analyze the railway system dependability. Saf. Sci. 110, 313–323.

Song, H., Liu, J., Schnieder, E., 2017. Validation, verification and evaluation of a train to train distance measurement system by means of colored petri nets. Reliab. Eng. Syst. Saf. 164, 10–23.

Valadares, D.C.G., Sobrinho, Á.A.D.C.C., Perkusich, A., Gorgonio, K.C., 2021. Formal verification of a trusted execution environment-based architecture for IoT applications. IEEE Internet Things J. 8, 17199–17210.

Valmari, A., 1996. The state explosion problem. In: Proceedings of the Advanced Course on Petri Nets, pp. 429–528.

Wang, R., Zheng, W., Liang, C., Tang, T., 2016. An integrated hazard identification method based on the hierarchical Colored Petri Net. Saf. Sci. 88, 166–179.

Wu, D., Zheng, W., 2018. Formal model-based quantitative safety analysis using timed coloured Petri nets. Reliab. Eng. Syst. Saf. 176, 62–79.

Wu, D. Verifiable design of a satellite-based train control system with petri nets. Dissertation, Braunschweig, Technische Universität Braunschweig, 2014.

Zhang, T., Li, X., Wu, D., Wang, H., Liu, J., Zhang, D., 2022. Evaluating the safety control scheme of railway centralized traffic control (CTC) system with coloured Petri nets. Sustainability 14, 11669.

Dr. Nazakat Ali is a Post-Doctoral Researcher in Dependable Software Engineering group at Mälardalen University, Department of Innovation, Design and Engineering in Västerås - Sweden. He has done his PhD from Chungbuk National University, Cheongju, South Korea in 2021. He received outstanding graduate researcher award during his PhD. Along with this, he is also a recipient of several best conference paper awards. During his PhD, Nazakat was working on the safety for collaborative cyber-physical systems. Before joining MDU, he was postdoctoral researcher at Software Intelligence Engineering Lab. Chungbuk National University, Cheongju South Korea during March 2021∼ June 2022, where he was working on learning-based safety analysis for supporting real-time collaboration in intelligent cyber-physical systems.

Sasikumar Punnekkat (Senior Member, IEEE) received the M.Tech. degree (Hons.) in computer science from the Indian Statistical Institute, in 1984, and the D.Phil. degree in computer science from the University of York, in 1997. His D.Phil. dissertation was titled ‘‘fault-tolerant scheduling of real-time systems.’’ He started his career as a Scientist Engineer with the Indian Space Research Organization (ISRO) and made significant contributions to the software development and testing of satellite launch vehicles. He was a recipient of the Commonwealth Scholarship of the U.K. He continued with ISRO. He was the head of software testing and reliability, until 2004, when he joined Mälardalen University, Sweden, where he has been the Chair of dependable software engineering, since 2007. He was also the Director of BITS, Goa campus, India, from 2015 to 2016. His research interests include multiple aspects of real-time systems, dependability, and software engineering. He has over 160 research publications in international conferences and journals (including five best paper awards). He has been a member of several program committees and has played a lead role in several EU and national projects, such as DAIS, InSecTT, SafeCer, SafeCoP, SUCCESS, EuroWeb, EURECA, FORA, Retnet, Progress, and Synopsis.

Abdul Rauf has been in the IT industry from last 15 years and has worked in different aspects of the product development life cycle including requirements, implementation, testing, and verification. This depth combined with the breadth of working experience in research and academic domains makes him quite flexible. He has focused on quality assurance and primarily worked in Test Management, Test Planning, Test Processes, Test Strategy, Test Design, Test Analysis, Manual Testing, Automation Testing, Regression Testing, Agile Testing, and Acceptance testing. He has performed PhD studies within the area of Test Coverage Analysis for GUI systems. Rauf is extremely interested in using artificial intelligence for solving Software Engineering Problems especially those related to Software Testing and verification.