THESIS

RESILIENCY ANALYSIS OF MISSION-CRITICAL SYSTEMS USING FORMAL METHODS

Submitted by

Mahmoud A. Abdelgawad

Department of Computer Science

In partial fulfillment of the requirements

For the Degree of Master of Science

Colorado State University

Fort Collins, Colorado

Spring 2025

Committee:

Advisor: Indrakshi Ray

Yashwant Malaiya
Sarath Sreedharan
Jeremy Daily

ABSTRACT

RESILIENCY ANALYSIS OF MISSION-CRITICAL SYSTEMS USING FORMAL METHODS

Mission-critical systems, such as navigational spacecraft and drone surveillance systems, play a crucial role in a nation's safety and security. These systems consist of heterogeneous systems that work together to accomplish critical missions. However, they are susceptible to cyberattacks and physical incidents that can have devastating consequences. To ensure resilience, missions must be designed so that mission-critical systems can withstand adverse events and continue to operate effectively, even with the occurrence of adverse events. In other words, engineers must specify, analyze, and anticipate potential threats, identify where adverse events may occur, and develop mitigation strategies before deploying a mission-critical system. This work presents an end-to-end methodology for analyzing the resiliency of critical missions. The methodology first specifies a mission in the form of a workflow. The mission workflow is then converted into a formal representation using Colored Petri Nets (CPN). Threat models are also extracted from the mission specification to tackle the CPN mission with various attack scenarios. These threat models are represented as CPN attacks. The methodology exploits the state transitions of the CPN mission attached to CPN attacks to analyze the resiliency of the mission. The analysis identifies which states the mission succeeds, fails, and is incomplete. The methodology is applied to a formation of mission-critical systems consisting of a military vehicle and route reconnaissance drones that work together to monitor a national border and respond immediately to physical threats. The result demonstrates how to restrict the mission to improve the resiliency of this formation. The result shows that this methodology is important for early-stage resiliency analysis to design resilient missions for mission-critical systems.

ACKNOWLEDGEMENTS

I take this opportunity to express my sincere gratitude to my advisor Professor Indrakshi Ray for her advice, support, and encouragement during my Master's program. Professor Ray's expertise and exceptional insights significantly contributed to the success of my work. I extend my gratitude to Professor Indrajit Ray for his valuable guidance during weekly meetings of the Cybersecurity Center Group (DBSec Group) here at Colorado State University.

I thank Professor Yashwant Malaiya, Professor Sarath Sreedharan, and Professor Jeremy Daily for their time and dedication in serving as members of my master's advisory committee. I thank all the teachers whose encouragement motivated me to strive for excellence.

I thank my friends and colleagues in the DBSec Group for their continuous support and encouragement, which have been instrumental in helping me achieve this goal.

Finally, I thank the Cybersecurity Center funded by the State of Colorado and the funding agencies for supporting this work. This work was supported in part by NIST funding under Award Number 60NANB23D152 and NSF under Award Numbers CNS 2335687, DMS 2123761, CNS 1822118, NIST, ARL, Statnett, AMI, NewPush and Cyber Risk Research.

DEDICATION

I dedicate this thesis to my wife, Aml, and my dear son, Attia—your love and patience have been my guiding light throughout this challenging and rewarding journey. This achievement is as much yours as it is mine.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

Chapter 1

Introduction

A mission-critical system is the formation of heterogeneous systems that interact to achieve a global goal [12]. These systems operate independently, but none can accomplish the goal on its own; thus, they work together towards broader objectives. The characteristics of mission-critical systems include being independently managed, geographically distributed, evolving to meet mission needs, emerging from a combination of behaviors and properties of system elements, and being influenced by a stimulus environment [12, 9].

Mission-critical systems have become common in technology domains, including spacecraft navigation systems and drone surveillance systems for military purposes. Standardization [16] also considers System of Systems (SoS), such as transportation, water and energy resources, smart grids, and smart cities, as mission-critical systems.

This work is confined to the domains of mission-critical systems in defense, where in an operational environment, the objectives are established based on critical missions [33, 20, 13]. Most military formations of mission-critical systems, such as ships, aircraft, satellites, and ground vehicles, are equipped with independent system elements (i.e., sensors, weapons, communications) that must fulfill mission objectives. Mission-critical systems are defined as systems whose failure of one system element can significantly impact mission-related functions [15, 26]. However, these systems are susceptible to adverse cyber events and physical incidents, and they must fulfill survivability requirements to continue a mission in the face of attacks [8].

Hence, a resilience analysis of critical missions is essential. In cyber space, resiliency is conceptually identical to the cyber survival defined for warfighter systems [28]. It focuses on evaluating cyber resilience goals, including anticipating, withstanding, continuing, recovering from, and adapting to adverse cyber events that could impact mission-related functions. Therefore, mission-critical system engineers must specify and analyze missions before deployment to assess their resilience and gauge what failures can be tolerated.

A mission can be described as a workflow consisting of various tasks executed by subjects and connected via different types of control flow operators [35]. The mission also has a set of objectives. If all objectives are satisfied, then the mission has *succeeded*. If some, but not all, objectives are satisfied, the mission is *incomplete*. When no objectives have been satisfied, the mission has *failed*.

1.1 A Challenge to Be Addressed

The literature on mission resiliency focuses primarily on handling subject unavailability as reassigning tasks based on subject attributes that provide them with the capability to perform tasks [21, 24, 32]. However, real-world adversarial scenarios of mission-critical systems extend beyond subject unavailability. Attackers may degrade system performance rather than completely remove components from the service. This performance degradation may cause a mission to abort or fulfill only a subset of its objectives. In other words, attackers do not necessarily remove notions of the system from service permanently; they degrade its capability, but this system may be able to continue to complete the mission. This requires a thorough analysis of mission capability and mitigation to reduce the impact of adversarial events.

Many types of workflow resilience are defined: static, one-shot, decremental, and dynamic resilience [32]. Static resiliency refers to a situation in which subjects become unavailable before the workflow executes and no subjects may become available during the execution. One-shot resiliency refers to the case when only one subject becomes unavailable during execution. Decremental resiliency expresses a situation where subjects become unavailable before or during the execution of the workflow, and no previously unavailable subjects may become available during execution. This work addresses the decremental resilience. Note that decremental resiliency implies one-shot and static resiliency. Manually analyzing workflows to verify resiliency is error-prone and labor intensive. Toward this end, there is a need for automated analysis of a mission and evaluation of potential attacks and their effects.

1.2 The Proposed Solution

To address this challenge, we propose a systematic methodology that integrates workflow modeling, Coloured Petri Nets (CPNs), and resiliency analysis technique to enhance mission continuity under adversarial conditions. The methodology consists of four key phases:

- **Workflow Modeling for Mission-Critical Systems**
  - Define a structured workflow representation that captures tasks dependencies and constraints.
  - Ensure a clear understanding of the task execution order and failure points to support mission resilience.
- **Transformation of Mission Workflow into Coloured Petri Nets (CPNs)**
  - Convert the high-level mission workflow into a CPN model for simulation and formal verification.
  - Encode task dependencies and constraints to analyze mission behavior under disruptions.
- **Resiliency Analysis of Mission Workflows**
  - Simulate adversarial attacks and system failures to assess mission resilience.
  - Identify vulnerabilities and response strategies to improve mission resilience against degrading threats.
- **Evaluation of the Methodology's Effectiveness**
  - Apply the proposed resilience methodology to real-world mission scenarios.
  - Assess the impact of resilience methodology in ensuring mission continuity under disruptions.

The methodology provides a systematic way for resiliency analysis to identify mission vulnerabilities and improve its resilience in adversarial environments.

1.3 Research Questions

- **RQ1:** How can workflows be systematically defined for mission-critical systems to model task dependencies and constraints requirements?
- **RQ2:** What transformations can be applied to convert high-level mission descriptions into Coloured Petri Nets for analysis and verification?
- **RQ3:** Is there a structured and scalable approach to analyzing the resilience of the mission under adversarial conditions and unexpected failures?
- **RQ4:** To what extent does the proposed resiliency analysis solution improve mission continuity and reduce the impact of disruptions on mission workflows?

These research questions guide the development, analysis, and validation of the resilience methodology proposed for mission-critical systems. We proceed with this thesis as follows. Chapter 2 introduces foundational concepts, including definitions and types of resilience, workflow modeling, and Coloured Petri Nets (CPNs) used for formal verification. Chapter 3 outlines the step-by-step methodology for modeling, transforming, and analyzing mission workflows under adversarial conditions. It also describes a running example. Chapter 4 demonstrates the application of the methodology to a realistic scenario involving a formation of mission-critical systems, converting mission specifications into formal representations and analyzing various attack scenarios. Chapter 5 interprets the findings, evaluates the effectiveness of the proposed methodology, and discusses contributions, limitations, and directions for future research. Chapter 6 summarizes related work on workflow resilience, Petri Net-based mission verification, and quantitative resilience assessment. Chapter 7 concludes the thesis by summarizing key insights and presenting future research avenues to improve a resiliency analysis for mission-critical systems.

Chapter 2

Background

This chapter establishes a solid foundation in key concepts and definitions that support the analysis and improvement of mission resilience. It first defines resilience and categorizes its various types: static, one-shot, decremental, and dynamic resilience. These classifications help in understanding how systems can withstand, continue, recover, and adapt under different failure conditions. Next, it introduces the concept of workflows, which are structured sequences of tasks designed to achieve mission objectives. Workflows are essential for modeling task execution, dependencies, and constraints in mission-critical environments. Finally, the chapter provides an overview of Coloured Petri Nets (CPN), a formal modeling technique used to analyze resilience. CPN enable structured verification and simulation of critical missions under adversarial conditions.

2.1 Resilience Definition & Types

Resilience, as defined by NIST Special Publication 800-160, Volume 2, is the ability of a system to anticipate, withstand, recover from, and adapt to adverse conditions, stresses, attacks, or compromises [28]. Unlike traditional security approaches that focus solely on prevention, resilience emphasizes continued functionality even under persistent threats and failures. This concept applies to both the cyber and physical space of systems. It ensures that critical infrastructure, organizations, and mission-critical operations remain operational despite adversarial disruptions. NIST emphasizes that resilience is an engineering discipline that integrates multiple layers of defense, redundancy, and adaptability across an organization's technical and operational domains.

For mission-critical systems, resilience ensures that even if a system fails or is compromised, the overall mission-critical systems can reconfigure, reroute, and shift operational priorities without catastrophic failure. This makes resilience a core requirement for national security, defense, space missions, and industrial control systems [28].

In terms of workflow, several resilience models have been defined in the literature [7, 35, 36], with varying degrees of adaptability to user unavailability and changes in task execution. The three primary types of resilience are static, decremental, and dynamic resilience.

Static Resilience: A workflow is *statically resilient* if it can be successfully completed even when a predetermined subset of users is unavailable before execution starts. Once the workflow begins, no additional users become unavailable, and those unavailable before execution remain so. Static resilience is in *co-NP complexity*, making it challenging but feasible to analyze using constraint-based methods.

One-Shot Resilience: A workflow is *one-shot resilient* if a single disruption event occurs during execution, causing some users to become unavailable, but the workflow can still be completed. Only one instance of user removal is allowed, and once a user is removed, they do not return. One-shot resilience is more flexible than *static resilience* but more restrictive than *decremental resilience*.

Decremental Resilience: A workflow is *decrementally resilient* if users can become unavailable before or during execution, but once removed, they do not return. Unlike static resilience, disruptions can occur at any stage during execution. However, removed users do not regain availability. Decremental resilience subsumes static resilience since it accounts for both pre-execution and runtime unavailability. Deciding decremental resilience is *PSPACE-complete*, making it computationally more complex than static resilience.

Dynamic Resilience: A workflow is *dynamically resilient* if users can become unavailable and available again at any point before or during execution. Dynamic resilience models real-world scenarios where personnel availability fluctuates due to external conditions, requiring adaptive resource allocation. No exact approach has been established to guarantee dynamic workflow resilience under all conditions, making it an open research problem.

The hierarchical relationship among resilience types is as follows:

$$\text{Dynamic Resilience} \supset \text{Decremental Resilience} \supset \text{One-Shot Resilience} \supset \text{Static Resilience}$$

Dynamic resilience is the most flexible, but computationally challenging to verify. Decremental resilience offers moderate adaptability, accounting for disruptions, but without recovery mechanisms. Static resilience is the most restrictive, requiring workflows to be planned based on a fixed availability assumption.

2.2 Workflow Definition

Typically, a workflow consists of tasks connected through operators [31, 31, 4, 6, 7, 35]. The syntax of the workflow adapted from [35] is defined as follows.

Definition 1 (Workflow)**.** *A workflow is defined recursively as follows.*

$W = t_i \otimes (t | W_1 \otimes W_2 | W_1 \# W_2 | W_1 \& W_2 | if\{C\}\, W_1\, else\, W_2 | while\{C\}\{W_1\}) \otimes t_f$

where

- t *is a user-defined atomic task.*
- t_i *and* t_f *are unique initial tasks and unique final tasks, respectively.*
- $\otimes$ *denotes the sequence operator.* $W_1 \otimes W_2$ *specifies* W_2 *is executed after* W_1 *completes.*
- $\#$ *denotes the exclusive choice operator.* $W_1 \# W_2$ *specifies that either* W_1 *executes or* W_2 *executes but not both.*
- $\&$ *denotes the and operator.* $W_1 \& W_2$ *specifies that both* W_1 *and* W_2 *must finish executing before the next task can start.*
- $if\{C\}\, W_1\, else\, W_2$ *denotes the conditioning operator.* C *is a Boolean expression. Either* W_1 *or* W_2 *execute based on the result of evaluating* C *but not both.*
- $while\{C\}\{W_1\}$ *denotes iteration operator. If* C *evaluates to true,* W_1 *executes repeatedly until expression* C *evaluates to false.*

Definition 2 (Simple and Complex Operators)**.** *A simple operator is an operator that imposes one single direct-precedence constraint between two tasks. The only simple operator is the sequence operator. All other operators are referred to as complex operators.*

Definition 3 (Simple and Compound Workflows)**.** *A simple workflow is a workflow that consists of at most one single complex operator and finitely many simple operators. All other workflows are compound workflows. A compound workflow can be decomposed into component workflows, each of which may be simple or compound.*

Definition 4 (Entities)**.** *The tasks of a workflow are executed by active entities, referred to as subjects. The entities in which we perform tasks are referred to as objects. An entity has a set of typed variables that represent its attributes.*

Definition 5 (State of an Entity)**.** *The values of the attributes of an entity constitute its state. The state of a subject determines whether it can execute a given task. The state of an object determines whether a task can be performed on it.*

In the remainder of this work, we abstract from objects and only consider critical systems specified as subjects.

Definition 6 (Mission)**.** *A mission is expressed as a workflow with a control-flow, a set of subjects, a subject to task assignment relation, and some initial conditions and objectives. Formally,* $\mathcal{M} = (W, \mathcal{S}, \mathcal{ST}, I, O)$ *where* W *is the control flow corresponding to the mission,* $\mathcal{S}$ *is a set of subjects,* $\mathcal{ST} \subseteq S \times Tasks(W)$ *is the set of subject-to-task assignments,* I *are the initial set conditions and* O *is the set of mission objectives. The subject of assignment of tasks should satisfy the access control policies of the mission. The conditions and objectives are expressed in predicate logic.*

2.3 Coloured Petri Nets (CPN)

A Colored Petri Net (CPN) is a directed bipartite graph, where the nodes correspond to the places P and transitions T. Arcs A are directed edges from a place to a transition or a transition to a place. The input place of transition is a place for which a directed arc exists between the place and the transition. The set of all input places for a transition $r \in T$ is denoted as $\bullet r$. An output place of transition is a place for which a directed arc exists between the transition and the place. The set of all output places of a transition $r \in T$ is denoted as $r\bullet$. Note that we distinguish between

tasks and transitions using the label r for transitions. CPNs operate on multisets of typed objects called tokens. The places are assigned tokens at initialization. Transitions consume tokens from their input places, perform some action, and output tokens on their output places. Transitions may create and destroy tokens through their executions. The distribution of tokens over the places of the CPN defines the state, referred to as marking, of the CPN. Formally, a Non-Hierarchical CPN is defined [19, 18] as $CPN = (P,T,A,\Sigma,V,C,G,E,I)$, where P, T, A, Σ, and V, are sets of places, transitions, arcs, colors, and variables, respectively. C, G, E, I are functions that assign colors to places, guard expressions to transitions, arc expressions to arcs, tokens at initialization expression, respectively.

Definition 7 (Simple Workflow CPN)**.** *A simple workflow CPN is a CPN that models a simple workflow. A simple workflow CPN has a unique input place i and a unique output place o.*

Definition 8 (CPN Module)**.** *A CPN Module consists of a CPN, a set of substitution transitions, a set of port places, and a port type assignment function. The set of port places defines the interface through which a module exchanges tokens with other modules. Formally a CPN module is defined as in [18] as:* $CPN_M = (CPN, T_{sub}, P_{port}, PT)$ *where (i) CPN is a Colored Petri Net (ii)* $T_{sub} \subseteq T$ *is a set of substitution transitions (iii)* $P_{port} \subseteq P$ *is a set of port places (iv)* $PT : P_{port} \rightarrow \{IN, OUT, IN \backslash OUT\}$

Definition 9 (Simple CPN Module)**.** *A Simple CPN Module is a CPN module in which the CPN is a Simple Workflow CPN and the interface of the module is defined by a single input port* i' *and a single output port place* o'*.*

Definition 10 (Module Hierarchy)**.** *A Module Hierarchy is a directed graph where each module* $s \in S$ *is a node; and for any two modules* $s_1, s_2 \in S$*, there exists a directed arc from* s_1 *to* s_2 *if and only if there is a substitution transition in* s_1 *that is mapped to module* s_2*. As represented in [18], the module hierarchy is formally defined as:* $MH = (N_{MH}, A_{MH})$ *where (i)* $N_{MH} = S$ *is the set of nodes, and (ii)* $A_{MH} = \{(s_1, r, s_2) \in N_{MH} \times T_{sub} \times N_{MH} \mid r \in T_{sub}^{s_1} \wedge s_2 = SM(r)\}$

A module with no incoming arcs in the module hierarchy is refereed to as a prime module.

Chapter 3

Resiliency Analysis Methodology

The methodology is designed with an abstract and adaptable structure to ensure its applicability across various mission-critical system domains. This flexibility allows it to be tailored to different operational environments while maintaining its core principles of resiliency analysis.

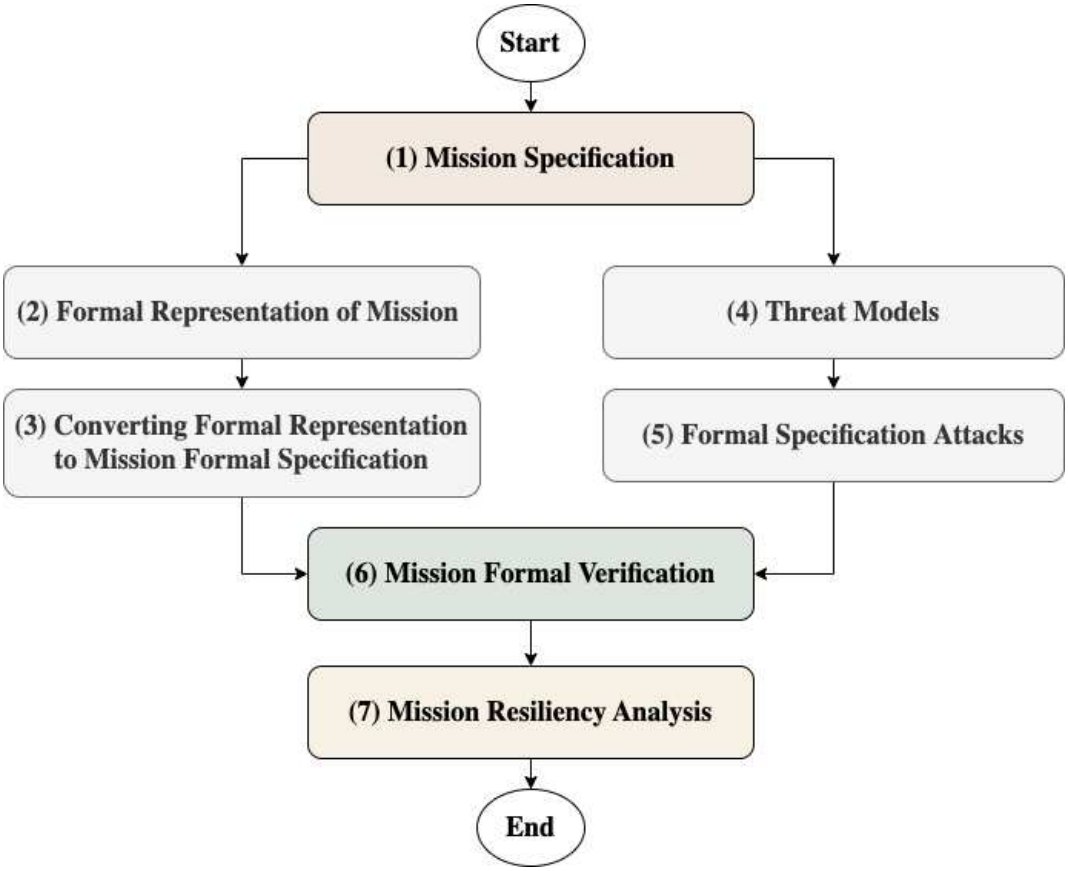


Figure 3.1: Methodology for Mission Resiliency Analysis

As illustrated in Figure 3.1, the methodology consists of 7 steps, described below. It starts with a mission specification description as input and ends with a mission resiliency analysis. Step 1 is essential because the next steps depend on its outcome. If a portion of the mission specification is overlooked, it will not be covered in the following steps. Steps 2 and 4 obtain the output from

Step 1, process it, and pass the result on to Steps 3 and 5, respectively. Steps 2 and 3 are processed sequentially, as one step's output is the input for the next step. Similarly, steps 4 and 5 are processed sequentially. The results of Steps 3 and 5 are integrated into the process of Step 6. This integration is analyzed in Step 7. The 7 steps are described below.

Step 1 - Mission Specification: It defines mission *subjects* as active entities that perform tasks. Each *subject* has a type and a set of variables corresponding to its *attributes*. The values of the attributes constitute a subject state. The state of a subject determines whether it can execute a given task. Step 1 also defines a set of *tasks*, which are atomic units of the mission. The mission starts with an initial task and ends with a final task. A mission includes a set of intermediate tasks between the initial and final tasks that may be grouped into subsets of tasks, referred to as sub-workflow in the formal representation. A mission also has a set of *objectives* to be accomplished. If all *objectives* are achieved, the mission has *succeeded*. If some of *objectives* are achieved but not all of them, the mission is *incomplete*. When no objectives are satisfied, the mission is *failed*.

Step 2 - Formal Representation of Mission: This step represents the mission as a workflow. It decomposes the mission into many sub-workflows and connects them into a main workflow to represent the entire mission. The workflow is then instantiated by initializing each subject and its attributes, assigning each subject to tasks, and assigning initial conditions to subjects and objectives.

Step 3 - Converting Formal Representation to Mission Formal Specification: The mission workflow is transformed into a formal specification to verify correctness and analyze its resiliency.

Step 4 - Mission Threat Models: The threat models are derived from the mission specification. In this work, we focus on threats that may change the values of a subject's mutable attributes. When a subject's attribute value changes, it impacts the mission.

Step 5 - Formal Specification of Attacks: Each attack changes the mutable values of a subject. An attack requires preconditions to be satisfied for it to happen. The effect of the attack is described as a post-condition. Thus, each attack is specified with a precondition and a postcondition.

Step 6 - Formal Analysis of the Mission: This step combines the CPN of the mission with the CPN of the attack. The attack CPN can be attached only to the places in the mission CPN where the attack CPN precondition is satisfied. The state space and state transitions are verified and analyzed for each attack scenario attached to the mission formal specification.

Step 7 - Mission Resiliency Analysis: A mission resiliency can be analyzed, including one-shot, decremental, and dynamic resiliency [32]. Resiliency analysis highlights constraints restricting the mission to improve mission resiliency.

3.1 A Running Example

This section describes a mission-critical system as a formation of military vehicle and two rotary-winged drones. The military vehicle and drones exchange data through a Line Of Sight (LOS) communication. The military vehicle also exchanges data with an operations center site (i.e., a military base) through a Beyond Line Of Sight (BLOS) via satellite communication.

The mission requires the military vehicle to monitor the national border between the official ports of entry and respond promptly to any threat. We assume that the mission will occur in a region of interest with rough roads and terrain. Human elements, the military vehicle drivers (e.g., soldiers), usually plan several routes to reach a Point of Interest (POI). However, in many cases, it is challenging to anticipate the route hazards that they may encounter (e.g., driving down a steep hill). Thus, drones play the role of surveillance and reconnaissance for the military vehicle to keep a safe distance from dangerous situations. The drones fly 5 miles ahead of the military vehicle at low altitudes using the Global Positioning System (GPS) to inform locations. The drones accelerate and decelerate to keep a distance of 5 miles from the military vehicle. They are also equipped with

cameras and infrared sensors to transmit day and night visions. If the drones fly during the day, they will use the cameras; if it is night, they will use the infrared sensors. The drones continue to fly as long as the battery is sufficient to return to the military vehicle for recharging. We assume that the battery level contains 100 units, enough to fly for 4 hours. If the drone is 5 miles from the military vehicle, it needs 2 battery units to return to the vehicle for recharging. The military vehicle is estimated to take 30-45 minutes to reach the POI if human elements drive at average speed (60-70 miles per hour). The military vehicle usually reaches the POI, evaluates the situation, and returns to the starting point (i.e., station) without engagement. If the situation at the POI is evaluated for engagement, the human elements must receive instruction from the military base to engage (e.g., open fire against a threat). The mission requires the military vehicle to be fueled with a full tank of gasoline and connected to the military base. Activating the cameras and infrared sensors, each takes one battery unit. Before flying, the drone's status is checked (LOS communication is up, cameras and infrared sensors are on, and the battery is fully charged). The status of the military vehicle (full gasoline tank and connected to the military base) is also checked during deployment to accomplish a mission. The mission is successful if the military vehicle and drones reach the POI in time. It is also considered a success if the data collected by drones and human elements have been transmitted to the military base in real time and returned to the station safely.

Now, we need to analyze whether it is possible for the mission to succeed and in which attack scenarios the mission fails. An attack scenario changes the attributes of the military vehicle and drones (i.e., mutable attributes that can be manipulated) and renders this mission-critical system formation unable to perform a task. For instance, an attack scenario defuses the drones' cameras. It degrades its scanning capability, leaving the military vehicle unable to figure out the situation ahead, which may cause a mission to fail. Another attack scenario disrupts BLOS communication, rendering the military vehicle disconnected from the military base and unable to report the threat situation, which may cause an incomplete mission. The methodology systematically covers possible attack scenarios that cause degradation of mission-critical system functions, and each attack scenario must be analyzed to determine whether the mission can be successful, failed, or

incomplete with respect to the attack. A mission is resilient to an attack scenario if, after the attack occurs, it still meets its objectives.

The next chapter applies the methodology to this example, analyzes and verifies its resilience with various attack scenarios.

Chapter 4

Application & Analysis

4.1 Mission Specification to Formal Representation

This section applies the first two steps of the methodology to the formation of mission-critical systems described in Chapter 3, Section 3.1 to translate it into a formal specification.

We manually analyze the mission description to define the subjects, their attributes, tasks, and mission objectives. Only mutable attributes are considered. For instance, we assume that a human element is immutable; thus, this subject and its attributes are not considered mission specifications. The mission specification is summarized as follows.

- set of subjects = $\{vehicle, [drones]\}$
- set of *vehicle* attributes = $\{location, GPS, speed, fuel, LOS, BLOS\}$
- set of *drone* attributes = $\{location, GPS, fly, acceleration, battery, camera,$ *IR sensor*, $LOS\}$

The tasks are as follows.

1. check the *vehicle* status: $fuel$ is full, LOS_v communication is up, and *BLOS* communication is up.
2. check the $[drones]$ status: *battery* is set to 100 units, *camera* is on, *IR sensor* is on, and *LOS* communication is up.
3. drive the *vehicle* toward the POI with an average speed of 60-70 miles per hour.
4. fly $[drones]$ toward the POI and keep 5 miles distance from the $vehicle$.
5. if it is daytime, each $drone$ uses *camera* to scan the route and the POI area and provide day vision to the $vehicle$.

6. if it is nighttime, each *drone* uses *IR sensor* to scan the route and the POI area and provide night vision to the *vehicle*.

7. at the POI area, evaluate the situation, and report the military base in real-time.

8. if the threat requires engagement, the human element must receive instruction from the military base to engage.

9. the *vehicle* and $[drones]$ return to the station with data collected and evaluation reported.

The objectives = {*move toward POI*, *collect data*, *evaluate the situation*, *respond to threads*, *return to the station*}. Notice that we italicized the mission elements, which are used to formalize the mission representation.

4.1.1 Formal Representation of the Mission

We use the workflow definition illustrated in Chapter 2, Section 2.2, to represent a mission. The mission is divided into three sub-workflows. *Deployment* sub-workflow comprises check status and deployment tasks. *Moving-to-POI* sub-workflow consists of drive the vehicle, fly drones, keep 5-miles distance ahead, and scanning route tasks. *At-POI* sub-workflow includes evaluate situations, report to the base, and engaging tasks. The *Deployment*, *Moving-to-POI*, and *At-POI* sub-workflows, respectively, execute sequentially. These three sub-workflows connect to the main workflow that starts with the initial and final tasks. Each sub-workflow forms a set of tasks managed by operators. The operators are also used to connect sub-workflows to the main workflow. The complete workflow of the mission is described as follows:

$$W = init \otimes \{check\_status\,(vehicle) \otimes check\_status\,([drones]) \otimes deploy \otimes$$
$$\{if\,(daytime)\,\{turn\_on\,([cameras])\,else\{turn\_on\,([IR\ sensors])\} \otimes$$
$$fly\,([drones]) \otimes drive\_vehicle\} \otimes while\,(moving)\,\{keep\_distance\ \&\ scan\_route\}\} \otimes$$
$$if\,(locatio = POI)\,\{evaluate\_situation \otimes report\_base \otimes if\,(instruction)\,\{engage\}\} \otimes$$
$$return\_to\_base \otimes final$$

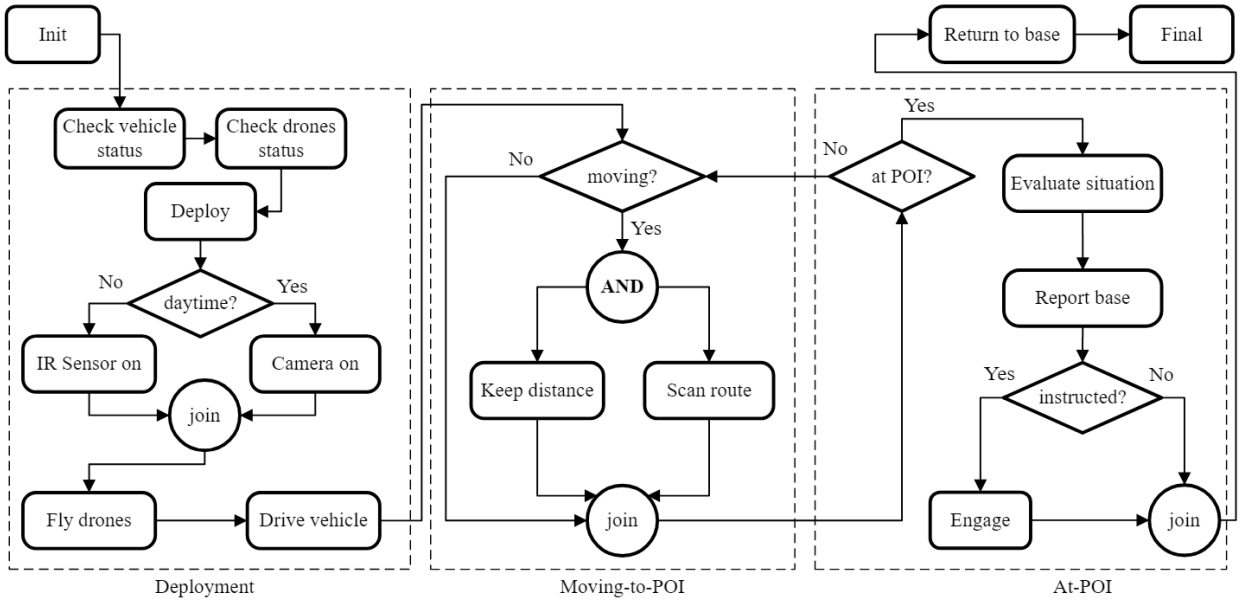


Figure 4.1: Mission Workflow

Figure 4.1 illustrates the workflow diagram for the mission. The dotted boxes represents the sub-workflows, *Deployment*, *Moving-to-POI*, and *At-POI*. The *return to base* task can be part of the main workflow since it does not represent any sub-workflow. Visualizing the workflow helps ensure the correctness of the workflow transformation to the CPN; hence, it is practical to draw the workflow.

Now, we instantiate the workflow. We initialize each subject and its attributes, assign each subject to a set of tasks, and set initial conditions for the subjects and objectives. There are two types of subjects *Vehicle* and *Drone*, and they are instantiated as $\mathcal{S} = \{vehicle : Vehicle;\ drones[] : Drone\}$. The subject type *Vehicle* has a set of attributes as $Vehicle.attributes = \{location : string;\ GPS : coordinate;\ speed \in < 0..240\};\ fuel \in \{0..80\};\ LOS : bool;\ BLOS : bool;\ evaluation\_reported : bool\}$. The subject type *Drone* has a set of attributes as $Drone.attributes = \{location : string;\ GPS : coordinate;\ fly : bool;\ acceleration \in 1..10\};\ battery \in \{1..100\};\ camera : bool;\ IR\_sensor : bool;\ LOS : bool;\ data\_collected : bool\}$. The subjects *vehicle* and $[drones]$ are assigned to tasks as $\mathcal{ST} = \{(vehicle, [drones], transition) \mid t \in Tasks(W)\}$.

Let I be a predicate logic formula giving the initialization conditions as:

$I = \exists\ s : S \mid (type = Vehicle) \wedge (location = \text{“}station\text{”}) \wedge (GPS = [34.0489, -111.0937]) \wedge$

$(speed = 0) \wedge (fuel = 80) \wedge (LOS = true) \wedge (BLOS = true) \wedge (evaluation\_reported = false)$
$\wedge (type = Drone) \wedge (location = \text{“}station\text{”}) \wedge (GPS = [34.0489, -111.0937]) \wedge (fly = false) \wedge$
$(acceleration = 0) \wedge (battery = 100) \wedge (camera = false) \wedge (IR\_sensor = false) \wedge$
$(data\_collected = false)$.

Let O be a predicate logic formula that defines mission objectives as follows:
$O = \exists\, s : S \mid (type = Vehicle) \wedge (type = Drone) \wedge (location = \text{“}station\text{”}) \wedge$
$(evaluation\_reported = true) \wedge (data\_collected = true)$.

We position the initial GPS coordinates for demonstration as [34.0489, -111.0937] and range the acceleration from 1 to 10. The GPS coordinate (i.e., Decimal Degrees (DD format) as [latitude, longitude]) is used to calculate the distance between the vehicle and the drones, and the acceleration is used to slow and speed up the drones to keep the 5-mile distance. As a result, the mission specification is described as follows:

$$\mathcal{M} = (W, \mathcal{S}, \mathcal{ST}, I, O)$$

4.1.2 Converting Formal Representation to Mission Formal Specification

We consider CPN as a formal model that is practical for expressing mission state transitions due to CPN providing color sets, a flexible way to define data types to represent various types of subjects [19, 18]. We use transformation rules defined in Chapter 2, Section 2.3 to convert the mission workflow tuple into a CPN tuple as:

$$\mathcal{M} = (W, \mathcal{S}, \mathcal{ST}, I, O) \xrightarrow[Rules]{Trans.} CPN = (P, T, A, \Sigma, V, C, G, E, I)$$

The transformation rules are briefly explained through the mission workflow:

Rule 1: Convert each task into a CPN transition.

Rule 2: Create input and output places for each transition, and create input and output arcs to pair the input place to the transition and the transition to the output place.

Rule 3: Produce a color set for each subject. The color set is a CPN datatype structure such as Product, Record, List, and Union [10]. The CPN color set combines primitive types such as

Bool, *Int*, *Real*, and *String*. It also can be a compound of primitive types and color sets. The color sets represent subject attributes. The color set that represents the Vehicle is declared as:

$color: \ Vehicle = \ \textbf{record} \ \{location : String; \ GPS : [Real, Real]; \ speed : Int \in \{1..240\}; \ fuel : Int \in \{1..80\}; \ LOS : Bool; \ BLOS : Bool; \ evaluation\_reported : Bool\}$

The color set that expresses the Drone is declared as:

$color: \ Drone = \ \textbf{record} \ \{location : String; \ GPS : [Real, Real]; \ fly : Bool; acceleration : Int \in \{1..10\}; \ battery : Int \in \{1..100\}; \ camera : Bool; IR\_sensor : Bool; \ LOS : Bool; \ data\_collected : Bool\}$

Rule 4: Assign the color sets to the CPN places and arcs to declare their datatypes and declares a set of variables such that there is a variable for each color set. For instance, the declaration *var vehicle* : *Vehicle* declares a variable *vehicle* of type *Vehicle*. The expression $colset \ droneList = list \ Drone \ with \ 2;$ declares a list of datatype *Drone* with maximum two elements, and the declaration *var drones* : *droneList* declares a variable *drones* of type *dronelist*.

Rule 5: Assign a guard expression for each CPN transition. The guard expression must be evaluated true in order for a transition to be executed (i.e., fired). For instance, $fly(drone_1) = true$ is a guard expression assigned to the deployment transition. It ensures that the deployment transition fires only if the $drone_1.fly$ attribute is true.

Rule 6: Assign an arc expression for each input and output arc of a transition. The arc expressions evaluate tokens to pass in and out from transitions (i.e., bindings). When a token is evaluated true by an arc expression, it enables the transition that connects to be fired. For instance, $if \ (at\_POI \ (vehicle) = true) \ \{vehicle\} \ else \ \{empty\}$ is an arc expression that ensures the *vehicle* token reaches the POI place so the token passes; otherwise, the POI place does not bind the next transition, and the next transition will not be executed.

Rule 7: Initialize variables with values and assign these variables to the CPN places as tokens. The initialization forms the initial conditions of the mission. The tokens are then manipulated in each CPN transition, generating the state transitions of the CPN. For instance, the initial conditions of the mission are described as:

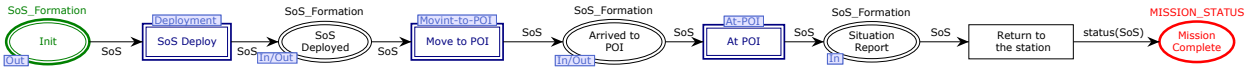
$vehicle = init\{(location = \text{“}station\text{”}) \wedge (GPS = [34.0489, -111.0937]) \wedge (speed = 0) \wedge$
$(fuel = 80) \wedge (LOS = true) \wedge (BLOS = true) \wedge (evaluation\_reported = false)\}$
$drone_1, drone_2 = init\{(location = \text{“}station\text{”}) \wedge (GPS = [34.0489, -111.0937]) \wedge$
$(fly = false) \wedge (acceleration = 0) \wedge (battery = 10) \wedge (camera = false) \wedge$
$(IR\_sensor = false) \wedge (data\_collected = false)\}.$

Figure 4.2 illustrates the *CPN* model resulting from applying the conversion rules 1-7 to the mission $\mathcal{M}$. The sub figure, 4.2 (a), represents the main CPN for the mission. The sub figures, 4.2 (b), (c), and (d), represent the sub CPNs for deployment, moving-to-POI, and At-POI respectively. We explicitly omitted writing each arc expression in the *CPN* model to maintain readability; see Appendix .1 for the CPN tool environment. In the CPN mission, we refer to the formation of a mission-critical system as *SoS*. The CPN transitions are coded using the Standard ML of New Jersey (SML/NJ [30]), which is used by the CPN tools ([27]), see Appendix .2.

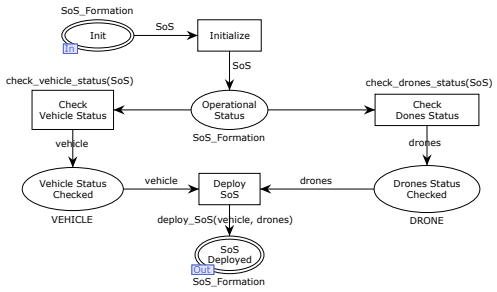
It shows the primary CPN mission, the mission main model, and three sub-CPN models: Deployment, Moving-to-POI, and At-POI. These three sub-CPNs are connected to the main CPN model through input and output ports, allowing the token to cross through the sub-CPN model and back to the main CPN. The small sub-CPNs, GPS attack, Battery, Camera, Sensor, LOS, and BLOS attack, depict CPN attack models, which are discussed in the next section.

4.2 Threat Model and Attacks Representation

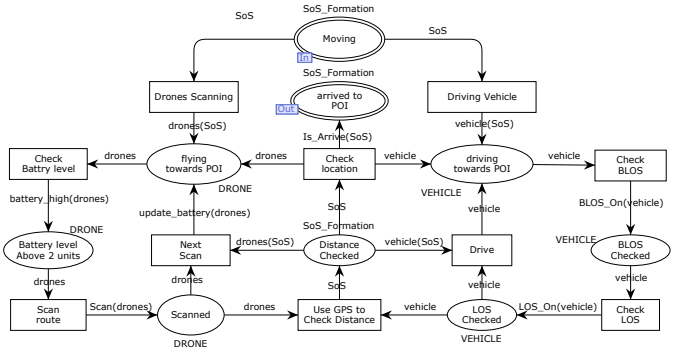
This section describes the threat models and attack representations for the mission.



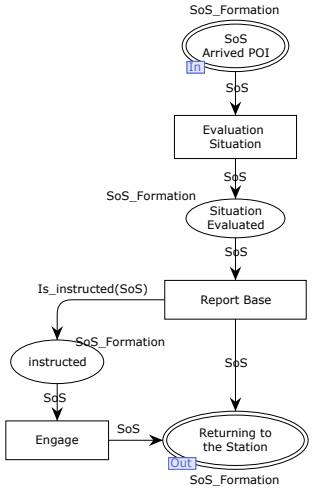
(a) Main CPN for the Mission



(b) Sub-CPN for Deployment



(c) Sub-CPN for Moving-to-POI



(d) Sub-CPN for At-POI

Figure 4.2: CPN Model of the Mission

4.2.1 Mission Threat Model

The threat model is derived directly from the mission description (Step 1). The vehicle and drones have mutable attributes that are attackable. We assume that drones can fly and accelerate are immutable, since they are physical attributes (i.e., mechanical attributes), thus excluding them from the threat model. Other subjects with entirely immutable attributes, such as human elements and satellites, are also omitted from the threat model. We consider *GPS*, *LOS*, and *BLOS* to be mutable attributes of the vehicle. We also consider *GPS*, *battery*, *camera*, *IR_sensor*, *LOS*, to be

mutable attributes of drones. Note that we focus on the communication and sensing capabilities of military vehicle and drones, as they are the best attackable attributes.

Table 4.1 illustrates the threat model for the mission. The GPS attack changes the appearance of where the vehicle is located. This leads the vehicle to drive in a different direction, unable to reach the POI, causing the mission to *fail*. It also dislocates the drones because the GPS calculates the distance between the vehicle and the drones. The drone's battery attack is considered to be consuming battery units.

Subject	Mutable Attribute	Attack Process	Impact on the Mission
Vehicle	*GPS*	Send out false signals to change the vehicle coordinates	The vehicle drives in the wrong direction, unable to reach the POI, leading to the mission to be *fail*. It may also cause the drones to hover away, unable to return, and the mission *fails*
	LOS	Disable the LOS communication	The vehicle loses communication with the drones, losing the route vision ahead; it may cause the mission to be *incomplete*
	BLOS	Disable the BLOS communication	The vehicle loses communication with the base, unable to receive instruction and unable to report the situation in real-time, leading the mission to be *failed*
Drone	*GPS*	Send out false signals to change the drones coordinates	The drones surveil incorrect route and collect useless data leading to the mission to be *incomplete*. They may also be unable to return to the vehicle and the mission *fails*
	battery	Consume one battery unit	It degrades the drones' ability to collect data and reduces flying time, leading to the mission to be *incomplete*; it might lead to mission *fails* if the drones do not have enough battery to return to the vehicle
	camera	Disable the camera	The drones are incapable of collecting data and cannot provide daytime vision to the vehicle, leading to the mission to be *incomplete*
	IR_sensor	Disable the camera	The drone is incapable of collecting data and cannot provide nighttime vision to the vehicle, leading to the mission to be *incomplete*
	LOS	Disable the LOS communication	The drone loses communication with the vehicle, unable to send the surveillance data to the vehicle, leading the mission to be *incomplete*

Table 4.1: Threat Model of the Mission

The battery consumption attack degrades the drone's capability for surveillance and reconnaissance, which requires more battery. It also reduces the flying time of the drones (e.g., instead of flying 4 hours in a regular situation, they will fly 3 hours when an attack occurs). If the battery

consumption attack occurs, we assume the drones should abort the mission and fly back to the vehicle, and the mission is incomplete. Another possibility is that the drones do not abort, keep flying, lose their battery, and then cannot return to the vehicle. In this case, the mission can fail because the data is not collected, and the vehicle cannot estimate the dangerous situations ahead. If the attacker turns off the camera and infrared sensor, the drones cannot collect data but can fly back to the vehicle, and the mission is incomplete. In addition to mission *succeeds*, two mission statuses are defined: mission is *incomplete* and mission *fails*. Mission success indicates that the drones collect data and fly back to the vehicle, and the vehicle reaches the POI and reports to the military base. The incomplete mission indicates that the drones cannot collect data, but can fly back to the vehicle, and the vehicle reaches the POI but cannot report to the military base. Mission failure demonstrates that drones cannot return to the vehicle (for example, they are lost), the vehicle cannot reach the POI, or the vehicle or drones do not return to the station. Analyzing the mission status from the mission specification is essential to translate attacks into formal specifications.

4.2.2 Formal Specification of Attacks

Each attack described in the threat model is formally specified as a CPN attack (that is, a sub-CPN model). The CPN attack is connected to the proper place of the CPN mission based on precondition and postcondition. From the threat model in Table 4.1, eight CPN attacks are formalized as three CPN attacks related to the vehicle, and five CPN attacks demonstrate drone attacks.

The three vehicle CPN attacks are presented in Figure 4.3 as vehicle GPS attack, vehicle LOS attack, and BLOS attack. These CPN attacks have the same precondition and postcondition as the *Vehicle* datatype. They must receive a variable of type *Vehicle*, change its values, and return a new state of this variable. These CPN attacks can be attached to any place in the CPN mission if that place has a precondition and postcondition of *Vehicle* datatype.

Similarly, the five CPN attacks for drones are formalized: drone GPS attack, battery attack, camera attack, infrared sensor attack, and drone LOS attack. These CPN attacks have precondition

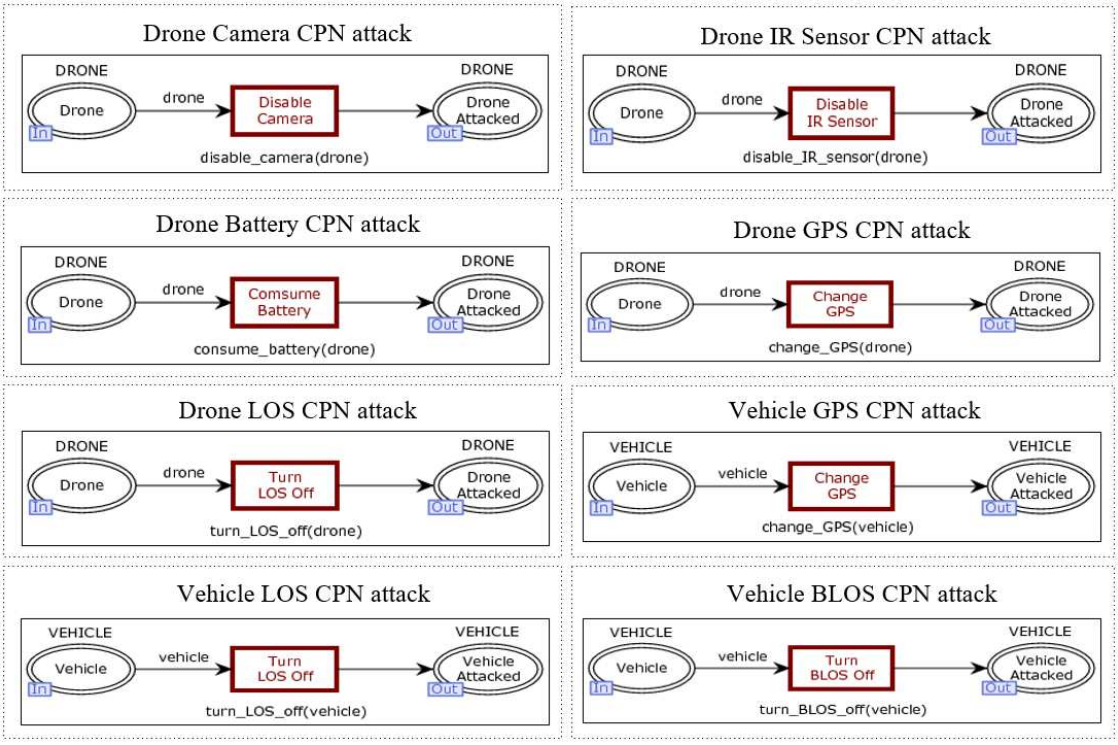


Figure 4.3: CPN Attacks for the Mission

and postcondition of datatype *Drone* and must receive a variable of type *Drone*, change its values and return the new state of this variable. For instance, the CPN battery attack decreases the value of the drone variable's attribute $drone_1.battery$ by 10 units and returns the drone variable with a new state of values. The CPN camera attack changes the value of the drone variable attribute $drone_1.camera$ to false, and the CPN infrared sensor attack changes the value of the drone variable attribute $drone_1.IR\_sensor$ to false and returns the drone with new values to the CPN mission. For these changes in the drone's attributes, a resiliency analysis is executed to verify where the CPN mission can succeed, be incomplete, or fail.

4.3 Mission Formal Specification Analysis

This section applies steps 6 and 7 of the methodology to analyze the mission resiliency. The state space and transitions of the CPN mission attached to the CPN attacks are verified using *CPNTools* [27].

We investigate the decremental resiliency of a mission in which a mission-critical system is degraded during execution and no previously unavailable capability may be recovered. We start with one attack that degrades the capability of a subject. We then decrease subjects' capabilities by different attacks and degrade them to demonstrate a decremental resiliency analysis. However, the number of possible attack scenarios is large. Each ordered sequence of CPN attacks produces an attack scenario; therefore, the number of attack scenarios generated from the 8 CPN attacks (shown in Figure 4.2) is factorial of 8 (that is, 8! = 40320 attack scenario). This large number of possible attack scenarios may lead to model state explosion of the CPN mission. As a result, several attack scenarios were selected to investigate decremental resiliency. We examine attack scenarios that stress the mission to be failed or incomplete. Analyzing stressful attack scenarios is more practical than others in assessing mission resiliency. Table 4.2 shows 4 attack scenarios that are used to demonstrate decremental resiliency.

Tasks	Location	Attack	Mission Status
Two drones scan the route and transmit to the military vehicle	Moving to POI	Disable cameras and IR sensors, and consume drones' battery	Incomplete
Two drones scan the route and transmit to the military vehicle	Moving to POI	Change GPS of the drones	Fail
The vehicle and drones are mapping the route and collecting data	Moving to POI	Disable LOS of the drones and the vehicle's BLOS	Fail
Report the situation to the base and wait for engaging instruction	At POI	Disable LOS, cameras and IR sensors of the drones and the vehicle's BLOS	Incomplete

Table 4.2: Decremental Attack Scenarios

The first attack scenario expresses that the two drones were attacked many times, turning off their cameras and IR sensors, and consuming their batteries with 10 units each. These attacks occurred while the mission-critical system formation was moving toward the POI. The drones could not collect data, but they had enough battery to safely return to the military vehicle using GPS. The mission status is incomplete because the data were never collected. The second attack scenario shows the drones' GPS being attacked while scanning the route, and they could not maintain the 5-mile distance. They may hover away from the military vehicle and get lost. The mission failed

because the drones did not return to the military vehicle. The third attack scenario also occurred while the mission-critical system formation was moving toward the POI. The attack disabled the vehicle's BLOS and the drones' LOS, rendering them unable to communicate. The drones could not transmit their GPS and data to the vehicle, nor could the vehicle communicate with the base. The mission failed because the drones could not reach the POI nor return to the vehicle. The fourth attack scenario happened at the POI location, where all communication and sensing capabilities of the vehicle and the drones were disabled. The mission was incomplete because the objectives of collecting data and reporting the base were not met, but the formation of the mission-critical system safely returned to the station.

4.3.1 State Space Analysis

The analysis examines the state space of the CPN model with each attack scenario. Table 4.3 illustrates the number of nodes and arcs of the state space model (that is, the reachability graph) generated for each attack scenario. It also shows the number of nodes and arcs of the strongly connected component (SCC) graphs generated from the state-space models. Table 4.3 reports that all CPNs of attack scenarios are full SCC. This indicates that the state-space models generated from these CPNs are acyclic; they do not have infinite loops.

However, it reports dead markings and dead transitions for each attack scenario, which are considerable for verification. Dead markings are states where the CPN is terminated, and dead transitions are those transitions that are not executed (no token visits them). Verifying dead markings signifies the values of the mission-critical system attributes where the mission is terminated, whereas verifying dead transitions signifies why the mission-critical system did not execute specific tasks.

The first attack scenario, moving to POI and disabling cameras and sensors and consuming battery, reports 4 dead markings ([30,34,37,42]) and 5 dead transitions. The state space of the second attack scenario, moving to POI and changing the drones' GPS, reports 3 dead markings ([25,26,31]) and 2 dead transitions. The third attack scenario, moving to POI and disabling drone

Location & Attack Scenario	State Space		SCC Graph		Status
Moving to POI, disable cameras and sensors, and consume battery	#Node 42	#Arcs 61	#Node 42	#Arcs 61	Full
	Dead Markings [30,34,37,42]		#Dead Transition 5		Live Transition None
Moving to POI, change GPS of the drones	#Node 31	#Arcs 38	#Node 31	#Arcs 38	Full
	Dead Markings [25,26,31]		#Dead Transition 2		Live Transition None
Moving to POI, disable LOS of the drones, and the vehicle's BLOS	#Node 37	#Arcs 54	#Node 37	#Arcs 54	Full
	Dead Markings [29,33,34,37]		#Dead Transition 5		Live Transition None
At POI, disable drones cameras, sensors, LOS, and vehicle's BLOS	#Node 63	#Arcs 85	#Node 63	#Arcs 85	Full
	Dead Markings [47,54,55,58,63]		#Dead Transition 7		Live Transition None

Table 4.3: State Space Report

LOS and vehicle BLOS, reports 4 dead markings ([29,33,34,37]) and 5 dead transitions. The fourth attack scenario, at POI that disables drone cameras, sensors, LOS and vehicle BLOS, reports 5 dead markings ([47,54,55,58,63]) and 7 dead transitions. Although the fourth attack scenario includes more attacks than the third attack scenario, the fourth attack scenario ends with an incomplete mission status compared to the third attack scenario, which terminates with a failed mission status. Interestingly, the state space of these attack scenarios reports an exact number of dead markings, only one node, and an exact number of dead transitions, 8 dead transitions. It turns out that the location where the attack scenario occurs significantly varies the mission results. Next, the dead markings and dead transitions are thoroughly verified and explained.

4.3.2 Formal Verification

We use the CPN-ML programming language [17] to verify the dead markings and the dead transition for each attack scenario. We also use built-in functions provided by the CPNTools [27], such as *Reachable(x,y)*, *AllReachable()*, *NodesInPath(x,y)*, *DeadMarking(x)*, *ListDeadMarkings()*, and *ListDeadTransitions()*, for the verification process. These functions return an execution sequence

for each dead marking in which the mission has succeeded, incomplete, or failed. We backtrack through the execution sequence and locate the state where the attack occurs. While backtracking, we verify the change of states in the sequence to inspect the reasons that lead to dead transitions and what state values the sequence terminates with at dead markings.

The state space of the first scenario includes three attacks in the Moving-to-POI location, which cause 4 dead markings and 5 dead transitions. The dead markings, States 30, 34, and 37, show that the CPN mission terminates where the drone's tokens loop over the battery check transition, which is attacked and ends with an empty mission status. State 42 expresses that the CPN mission terminates with mission status "Incomplete" because the value of $data\_collected$ is false and part of the mission objectives are not satisfied. The state space of the second attack scenario has dead markings, States 25 and 26, where the drones' GPS attacked in the Moving-to-POI location, which caused the CPN mission to terminate with an empty mission status. State 31 is another dead marking that the CPN mission completes the mission with mission status "Fail". Verifying this state shows that the drone's GPS does not equal the vehicle's GPS, indicating that the drone is not returning to the station. It also shows that the drone attribute $data\_collected$ is false and that mission objectives are not met. The two dead transitions are "$SoS'UseGPStoCheckDistance$ 1" and "$SoS'Checklocation$ 2", indicating that the GPS attack prevented the drone token from passing through these transitions. In the third attack scenario, the drone's LOS and the vehicle's BLOS were disabled in the Moving-to-POI location, causing 4 dead markings and 5 dead transitions. The dead markings, States 29, 33, and 34, show that the CPN mission cannot continue and terminates with an empty mission status. State 37 shows that the CPN mission ended with the mission state "Fail" because the drone's GPS differs from the vehicle's GPS due to disconnection, and the vehicle disconnected from the base. Verifying the 5 dead transitions shows that the vehicle BLOS check transition leads to check vehicle LOS, uses GPS to check the distance, and checks the location to be dead transitions. This is because these transitions are related to one another; failing one causes failing another. The impact of attacking LOS communication cascades to the GPS capability. The fourth attack scenario's state space generates 5 dead markings: States 47, 54, 55, 58, and

63. States 47, 54, 55, and 58 show that the CPN mission terminated with an empty mission status. These are due to the number of CPN attacks that disable cameras, sensors, LOS, and BLOS. State 63 shows that the CPN mission ended with the mission state "Incomplete". Although the number of attacks in the fourth scenario is more than the number of attacks in the third scenario, the status of the CPN mission ends with "Incomplete" compared to the third scenario that ended with "Fail". In the fourth attack scenario, the drone's GPS is the same as the vehicle's. The drones can collect data because the attacks occurred at the POI location, not the route. The CPN mission status in the fourth attack scenario is incomplete because the SoS cannot report the base; the BLOS is disconnected.

Verification shows where the mission fails in the attack dispute. It also points out that attacking the capability of one system affects other mission-critical system parties, and the mission fails. Lastly, the impact of attacks is different from one location to another (geographically); an attack is critical at one location and ineffective at another. Mission-critical system engineers must take these facts into account when describing resilient missions.

4.3.3 Mission Resiliency Analysis

The verification indicates that the formation of mission-critical system must immediately abort the mission in some instances, especially when LOS and BLOS communication is attacked. Continuous surveillance and movement toward POI without communication minimize the resilience level of the mission.

Task	Location	Restriction & Improvement
Drones scan the route and collect data	Moving to POI	Abort the mission if the drones' LOS are disabled. Provide recoverable LOS to pursue the mission
Drones scan the route and collect data	Moving to POI	Abort the mission if the drones' LOS are attacked. Or, provide recoverable LOS to pursue the mission
Drones scan the route and collect data	Moving to POI	Provide recoverable cameras and IR sensors for the drones as well as backup battery
Military vehicle moves toward POI	Moving to POI	Abort the mission if the vehicle' BLOS is attacked. Or, provide recoverable BLOS to communicate with the base
the formation reports POI situation to the base	At POI	Use backup communication channels when BLOS is attacked and provide recoverable BLOS for reporting

Table 4.4: Mission Restrictions & Improvements

To enhance the resilience of critical missions, appropriate restrictions must be counted on missions and improvements to formation capabilities must be considered. Table 4.4 expresses constraints and improvements to increase the level of resiliency of missions. That is, if we add these constraints and improvements to the mission described in Chapter 3, Section 3.1, this mission becomes resilient and resistant to various types of attacks.

Chapter 5

Discussion

This chapter analyzes the findings, emphasizes contributions, and situates the study within the broader research context.

5.1 Address the Research Questions

5.1.1 Define Workflows for Mission-Critical Systems

One of the primary research questions focuses on how workflows can be systematically defined for mission-critical systems to model task dependencies and constraints. The proposed methodology addresses this question by establishing a structured workflow representation that captures the dependencies between tasks and ensures a clear execution order. The findings indicate that the definition of workflows in the context of mission-critical systems enhances the analysis of mission continuity and resilience. Additionally, Colored Petri Nets (CPNs) play a key role in ensuring that mission workflows are both structured and verifiable under adversarial conditions.

5.1.2 Transform Mission Description into Coloured Petri Nets (CPNs)

The second research question investigates the transformation needed to convert a general mission description into Colored Petri Nets (CPN) for analysis and verification. The methodology presents for converting mission workflows into CPNs, which enables formal simulation and verification. The analysis indicates that CPN-based representations effectively allow us to evaluate mission behavior. This transformation plays a crucial role in assessing how missions respond to potential adversarial events.

5.1.3 Systematic Resiliency Analysis of Missions

Another core research question examines whether there is a structured approach to analyze the resilience of missions under adversarial events. The study demonstrates that the integration of

the CPN mission and CPN attacks provides a systematic solution to analyze mission resilience. Through resiliency analysis, potential attack scenarios are simulated to assess mission vulnerabilities. The results highlight the feasibility of automated mission resilience evaluation and reduce the labor intensiveness of manual analysis.

5.1.4 Effectiveness of the Proposed Resilience Methodology

The final research question examines how much the proposed resiliency analysis solution improves mission continuity. This study assesses the effectiveness of the methodology in identifying vulnerabilities and developing response strategies for critical missions. The findings indicate that resilience measures significantly enhance mission continuity by pinpointing weak points and suggesting potential countermeasures.

5.2 Implications and Contributions

The research provides several key contributions:

- A **systematic methodology** for modeling and analyzing mission resilience, which improves structured evaluation of mission-critical workflows.
- A **formal transformation approach** using CPNs that enables simulation and verification of mission execution under adversarial conditions.
- A **automated resilience analysis technique** that reduces human error and provides actionable insights into the vulnerabilities of the mission.
- **Validation through application**, demonstrating the effectiveness of the methodology in maintaining mission continuity despite disruptions.

5.3 Limitations and Future Work

While the methodology effectively enhances resilience analysis, there are areas for further improvement:

- **Computational Complexity**: CPN-based analysis can become computationally intensive for large-scale missions. Future work could explore optimization techniques to improve efficiency.

- **Dynamic Resilience Consideration**: The study focuses primarily on decremental resilience. Future research could extend the methodology to incorporate dynamic resilience, where subjects may recover over time.

- **Integration with AI-driven Predictive Analysis**: Resilience analysis could be augmented with AI-based prediction models to anticipate potential mission failures and adversarial attacks to improve critical missions.

5.4 Discussion Summary

The discussion highlights how the research questions are addressed through the proposed methodology and analysis. The study establishes a foundation for the evaluation of mission resilience using workflow modeling and CPN-based analysis. The findings demonstrate the effectiveness of the methodology in improving mission continuity and resilience against adversarial events. Future research could refine and extend this approach to include dynamic resilience and adaptive AI mission planning in real time.

Chapter 6

Literature Review

The literature addresses the workflow resiliency problem as the ability to complete a workflow, given that some users have become unavailable [29]. Some research defined various types of workflow resilience based on unavailability situations. Others addressed it as a satisfiability problem. Other researchers use Petri Nets to verify the workflow's correctness and analyze the workflow's resilience if a subject is permanently removed from the workflow, while others use quantitative models to measure the resiliency of the workflow.

6.1 Workflow Resilience

Wang *et al.* [32] introduce three types of resilience, static, decremental, and dynamic. They use the term user to refer to a subject, an active entity doing tasks. Static resilience refers to a situation in which subjects become unavailable before the workflow executes, and no subjects may become available during the execution. Decremental resiliency expresses a condition where subjects become unavailable before or during the execution of the workflow, and no previously unavailable subjects may become available during execution, while dynamic resilience describes the situation where a subject may become unavailable at any time; a previously unavailable subject may become available at any time. The different types of resilience formulation capture various types of attack scenarios.

The authors in [14] define one-shot resilience. Once-shot resilience describes a situation where a subset of users become unavailable before or during the execution of the workflow, but only once. Once a subset of users is unavailable, no other users become unavailable, and no user may become available again. One-shot resilience is a generalization of static resilience and a specialization of decremental resilience. It expresses the situation where a single subject removal is performed at an opportune time during execution rather than only before execution (static) or continuously throughout (decremental).

Zavatteri *et al.* [37] addresses the challenge of ensuring that workflows are executed despite uncertainty in resource availability. The author extends the Workflow Satisfiability Problem (WSP) to consider levels of workflow resiliency, introducing static, decremental, and dynamic resiliency as defined by Wang and Li [32]. The study examines how workflows can adapt in real-time to user unavailability, ensuring continued execution. The author develops a formal approach using Constraint Networks under Conditional Uncertainty (CNCUs) and applies controller synthesis techniques for workflow adaptation. The experiment demonstrates the enhancement of workflow resilience and makes workflows more robust to unexpected user absences.

6.2 Resilience as Satisfiability Problem

Yang *et al.* [35] address the workflow satisfiability problem to determine whether a set of subjects can complete a workflow. They investigated the computational complexity of the workflow satisfiability problem in two aspects. One aspect considers either one path or all paths in a workflow, and the other focuses on the possible patterns in a workflow. They present a set of algorithms to solve various types of workflow satisfiability problem. They show that many existential and universal workflow satisfiability problems are NP-complete and NP-hard. Thus, they conclude that restrictions on workflow patterns induce such problems to be solvable in polynomial time.

Crampton *et al.* [11] introduce the Bi-Objective Workflow Satisfiability Problem (BO-WSP) to address workflow satisfiability. The authors examine the computational complexity of BO-WSP from both the classical and parameterized complexity perspectives. The study focuses on user-independent constraints and demonstrates that computing a Pareto front for BO-WSP is fixed-parameter tractable (FPT). They present two algorithms, one based on a fixed parameter approach and another using mixed-integer programming (MIP), to solve BO-WSP. The experimental evaluation shows that the FPT-based algorithm is efficient. In addition, they explore workflow resiliency, proving that related decision problems are also FPT under user-independent constraints. They concluded that the BO-WSP model provides a practical means of optimizing workflow executions while ensuring computational efficiency.

Boughrous *et al.* [5] propose a workflow criticality-based approach to bypass the Workflow Satisfiability Problem (WSP) to ensure resilience in workflow management systems. The authors introduce workflow criticality as a metric to prevent WSP during run-time. They present a delegation process algorithm to utilize workflow criticality, delegation, and priority concepts. The approach extends previous work of representing user workload status and defines metrics for task priority and workflow criticality. The authors validate the approach through case studies, demonstrating its effectiveness in reducing workflow deadlock situations. They concluded that the method improves the resilience of the workflow.

6.3 Resilience using Petri Nets

Van der Aalst [31] represents a workflow as a structured process net. A process net is a Petri Net with unique input and output places. A process net is extended by adding transition and arcs connecting the output place to the input place, forming an extended process net that has to be strongly connected. The author verifies the correctness of the process net by checking whether the extended process net is live and bounded. He defines correctness as the absence of logical abnormalities such as deadlocks and live locks.

LI Xi-Zuo *et al.* [34] propose a Petri-Net-based reduction approach to verify the correctness of the workflow that represents business processes. The reduction approach addresses the complexity of Petri-net that corresponds to workflow models. The authors define a set of reduction rules, including merging serial places and transitions, removing linear-dependent place transitions, merging parallel places and transitions, and removing self-loop places and transitions.

6.4 Quantitative Resilience

Mace *et al.* [23, 22, 21] propose a quantitative measure of workflow resiliency. They used a Markov Decision Process (MDP) to model workflow to provide a quantitative measure of resilience. They refer to binary classification, such as returning an execution sequence if one exists

and declaring the workflow resilient or returning false and declaring the workflow not resilient. The authors show that MDP models give a termination rate and an expected termination step.

In [25], Mace *et al.* developed a tool to analyze workflow resiliency, the so-called introduce Workflow Resiliency Analysis and Design (WRAD). They define workflow resiliency as the probability of completing a workflow while satisfying constraints despite user unavailability. They formalize workflow resiliency using Markov decision processes (MDPs) and implement WRAD as an automated tool that encodes workflow descriptions into the PRISM probabilistic model checker. WRAD computes the current resiliency of a workflow and identifies optimal changes to constraints to ensure that a given resiliency threshold is met. The experiments show that WRAD enables workflow designers to balance productivity by proactively adjusting constraints. They concluded that WRAD provides a scalable and effective approach for workflow resiliency analysis before execution.

6.5 Research Gap

Our work [1, 2, 3] argues that the workflow resiliency problem can sometimes be viewed as unavailability and degradation. In other words, attacks do not permanently remove subjects from service; they decrease their capabilities. Consider the drone surveillance example; a failure in the drone's camera affects the termination and success of the workflow. Regarding resilience based on availability, one can assume whether the camera's loss is critical enough to remove the drone from the workflow. The drone should be kept, as its other sensors can complete the workflow.

We take advantage of other work for analyzing the workflow, using formal specifications, and investigating resiliency. We design the methodology to cover various aspects of mission resiliency, one-shot, and decremental resiliency. We differ from others in a systematic way of analyzing mission specifications, constructing mission workflow, generating threat models, and investigating various attack scenarios that exercise mission resiliency. This methodology provides reasonable improvements that increase the resiliency of missions for mission-critical systems.

Chapter 7

Conclusion and Future Work

This thesis introduces a methodology that addresses mission resiliency analysis for mission-critical systems. The methodology systematically analyzes various decremental resilience of missions in mission-critical systems. The methodology consists of seven steps, starting from a description of a mission, a formal representation of the mission, converting the mission's formal representation to formal specification, constructing threat models, formal specification of attacks, and ending with resiliency analysis. The methodology uses a workflow for the formal representation of the mission. It also constructs threat models and converts them into formal representations of attacks. The methodology then utilizes Coloured Petri Nets (CPN) to specify missions with attached attacks.

We applied the methodology to a mission-critical system formation demonstrating a military vehicle and two drones working together to monitor a national border. The results show that the resiliency of the mission is a degradation issue. When attacks occur, a mission-critical system cannot continue and the mission fails or is incomplete. The methodology indicates in which state the mission succeeds, fails, and is incomplete. It also highlights restrictions that must be added to the mission to improve resilience.

Future work will investigate the application of the methodology to dynamic resiliency, where a system is able to recover its components after an attack. The model state explosion will be investigated to verify such scenarios. Future work will also use threat modeling frameworks such as STRIDE and PASTA to cover various types of attacks that violate confidentiality, integrity, and availability in the context of mission-critical systems.

References

[1] Abdelgawad, M., Ray, I.: Methodology for resiliency analysis of mission-critical systems. In: Proceedings of the 39th ACM/SIGAPP Symposium on Applied Computing. pp. 1292–1300 (2024)

[2] Abdelgawad, M., Ray, I.: Resiliency analysis of mission-critical system of systems using formal methods. In: IFIP Annual Conference on Data and Applications Security and Privacy. pp. 153–170. Springer (2024)

[3] Abdelgawad, M., Ray, I., Vasquez, T.: Workflow resilience for mission critical systems. In: the proceedings of the 25th International Symposium of Distributed Systems Stabilization, Safety, and Security (SSS). vol. 14310, pp. 498–512. Springer, Jersey City, NJ, USA (2023)

[4] Arpinar, I.B., Halici, U., Arpinar, S., Doğaç, A.: Formalization of workflows and correctness issues in the presence of concurrency. Distrib. Parallel Databases **7**(2), 199–248 (apr 1999)

[5] Boughrous, M., El Bakkali, H.: A workflow criticality-based approach to bypass the workflow satisfiability problem. Security and Communication Networks **2021**(1), 3330923 (2021)

[6] Bride, H., Kouchnarenko, O., Peureux, F.: Verifying modal workflow specifications using constraint solving. In: the proceedings of the Integrated Formal Methods (IFM). pp. 171–186. Springer (2014)

[7] Bride, H., Kouchnarenko, O., Peureux, F., Voiron, G.: Workflow nets verification: SMT or CLP? In: the proceedings of the Critical Systems: Formal Methods and Automated Verification (FMICS-AVoCS). pp. 39–55. Springer (2016)

[8] Chong, J., Pal, P., Atigetchi, M., Rubel, P., Webber, F.: Survivability architecture of a mission critical system: The DPASA example. In: the proceeding of the 21st Annual Computer Security Applications Conference (ACSAC). pp. 495–504. IEEE, Tucson, AZ, USA (2005)

[9] Cook, S., Pratt, J.: Advances in systems of systems engineering foundations and methodologies. Australian Journal of Multi-Disciplinary Engineering **17**(1), 9–22 (2021)

[10] CPN Tools Administration: Help topics for verification. http://cpntools.org/2018/01/15/verification/ (jan 2018), online; accessed 5 January 2025

[11] Crampton, J., Gutin, G., Karapetyan, D., Watrigant, R.: The bi-objective workflow satisfiability problem and workflow resiliency 1. Journal of Computer Security **25**(1), 83–115 (2017)

[12] Dahmann, J., Roedler, G.: Systems of systems engineering standards. Insight **19**(3), 23–26 (2016)

[13] Figueira, N., Pochmann, P., Oliveira, A., de Freitas, E.P.: A C4ISR application on the swarm drones context in a low infrastructure scenario. In: the proceedings of the International Conference on Electrical, Computer and Energy Technologies (ICECET). pp. 1–7. IEEE, Prague, Czech Republic (2022)

[14] Fong, P.: Results in workflow resiliency: Complexity, new formulation, and ASP encoding. In: the proceedings of the Ninth ACM Conference on Data and Application Security and Privacy (CODASPY). pp. 185–196. ACM (2019)

[15] Houliotis, K., Oikonomidis, P., Charchalakis, P., Stipidis, E.: Mission-critical systems design framework. Advances in Science, Technology and Engineering Systems Journal **3**(2), 128–137 (2018)

[16] ISO/IEC/IEEE: International standard – systems and software engineering – system of systems (SoS) considerations in life cycle stages of a system. ISO/IEC/IEEE 21839:2019(E) **1**(8767114), 1–40 (2019)

[17] Jensen, K., Kristensen, L.M.: CPN ML Programming, pp. 43–77. Springer, Berlin, Heidelberg (2009)

[18] Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer, Berlin Heidelberg (2009)

[19] Jensen, K., Kristensen, L.M., Wells, L.: Coloured Petri Nets and CPN tools for modelling and validation of concurrent systems. International Journal on Software Tools for Technology Transfer **9**(3-4), 213–254 (2007)

[20] Lappas, V., Shin, H.S., Tsourdos, A., Lindgren, D., Bertrand, S., Marzat, J., Piet-Lahanier, H., Daramouskas, Y., Kostopoulos, V.: Autonomous unmanned heterogeneous vehicles for persistent monitoring. Drones **6**(4), 94 (2022)

[21] Mace, J., Morisset, C., van Moorsel, A.: Quantitative workflow resiliency. In: the proceedings of the Computer Security (ESORICS). vol. 8712, pp. 344–361. Springer, Wroclaw, Poland (2014)

[22] Mace, J., Morisset, C., van Moorsel, A.: Modelling user availability in workflow resiliency analysis. In: the proceedings of the 2015 Symposium and Bootcamp on the science of security (HotSoS). pp. 1–10. ACM, Urbana Illinois, USA (2015)

[23] Mace, J., Morisset, C., van Moorsel, A.: Resiliency variance in workflows with choice. In: the proceeding of the Software Engineering for Resilient Systems (SERENE). pp. 128–143. Springer, Paris, France (2015)

[24] Mace, J., Morisset, C., Moorsel, A.v.: Impact of policy design on workflow resiliency computation time. In: the proceedings of the Quantitative Evaluation of Systems (QEST). vol. 9259, pp. 244–259. Springer, Madrid, Spain (2015)

[25] Mace, J.C., Morisset, C., van Moorsel, A.: Wrad: Tool support for workflow resiliency analysis and design. In: the proceedings of the 8th International Workshop, Software Engineering for Resilient Systems (SERENE). pp. 79–87. Springer, Gothenburg, Sweden (2016)

[26] Ponsard, C., Massonet, P., Molderez, J.F., Rifaut, A., Lamsweerde, A.v., Van, H.T.: Early verification and validation of mission critical systems. Formal Methods in System Design **30**(3), 233–247 (2007)

[27] Ratzer, A.V., Wells, L., Lassen, H.M., Laursen, M., Qvortrup, J.F., Stissing, M.S., Westergaard, M., Christensen, S., Jensen, K.: CPN tools for editing, simulating and analysing coloured Petri nets. In: the proceeding of the 24th International Conference on Applications and Theory of Petri Nets (ICATPN). vol. 2679, pp. 450–462. Springer, Eindhoven, Netherlands (2003)

[28] Ross, R., Pillitteri, V., Graubart, R., Bodeau, D., McQuaid, R.: Nist special publication 800-160 volume 2: Developing cyber resilient systems. NIST Special Publication **2**, 224 (2019)

[29] Santos, D.R.d., Ranise, S.: A survey on workflow satisfiability, resiliency, and related problems. arXiv preprint arXiv:1706.07205 (2017)

[30] Standard Meta-Language (SML/NJ): The Standard ML of New Jersey (SML/NJ) (2024), https://www.smlnj.org, accessed: February 14, 2025

[31] Van Der Aalst, W.: Structural characterizations of sound workflow nets, Computing science reports, vol. 9623. Technische Universiteit Eindhoven (1996)

[32] Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM transactions on information and system security **13**(4), 1–35 (2010)

[33] Weisman, M., Kott, A., Ellis, J., Murphy, B., Parker, T., Smith, S., Vandekerckhove, J.: Quantitative measurement of cyber resilience: Modeling and experimentation. Computing Research Repository (CoRR) **ABS/2303.16307** (2023)

[34] Xi-zuo, L., Gui-ying, H., Sun-ho, K.: Applying petri-net-based reduction approach for verifying the correctness of workflow models. Wuhan University Journal of Natural Sciences **11**, 203–210 (01 2006)

[35] Yang, P., Xie, X., Ray, I., Lu, S.: Satisfiability analysis of workflows with control-flow patterns and authorization constraints. IEEE Transactions on Services Computing **7**(2), 237–251 (2014)

[36] Zavatteri, M., Viganò, L.: Last man standing: Static, decremental and dynamic resiliency via controller synthesis. Journal of computer security **27**(3), 343–373 (2019)

[37] Zavatteri, M., et al.: Temporal and resource controllability of workflows under uncertainty. In: BPM (PhD/Demos). pp. 9–14 (2019)

Appendices

.1 Appendix A: Colored Petri Nets Tool Environment

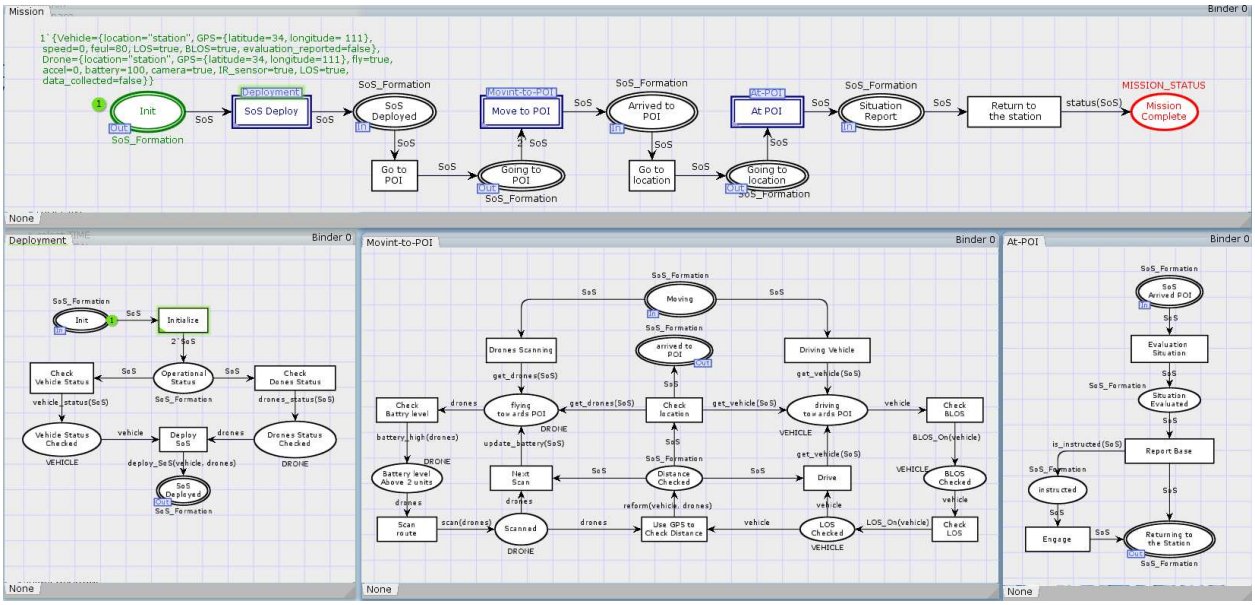
The CPN-tool version 4 [27] was used to model a mission-critical system formation (SoS) and the patrolling mission. The figure below shows 4 CPN nets described as follows.

1. Mission CPN net: is the main CPN net that represents the mission-critical system formation as a green token and the mission places and transitions. This net controls three CPN subnets: Deployment, Move-to-POI, and At-POI.

2. Deployment CPN subnet: represents the process of mission deployment.

3. Move-to-POI net: represents the formation moving toward the Point of Interest (POI).

4. At-POI net: represents the formation is at (POI).



.2 Appendix B: Source-code of CPN transitions Using SML

Standard Meta-Language (SML/NJ [30]) is used within the CPN tool to define state transition, update mechanism, and system behavior in a structured and functional programming paradigm. The SML code shown below captures the dynamic behavior of a mission-critical system formation (military vehicle and two drones). The primary objectives of these transitions include:

- **Assess system status:** evaluate key parameters (e.g., fuel levels, sensor availability, and communication links).
- **Deploying mission:** update system attributes dynamically.
- **Simulate attacks:** demonstrate degradation, e.g., loss of LOS service.
- **Verify mission continuity:** check operational parameters and mission status (success, incomplete, or fail).

```
(* Project: Decremental Resiliency Analysis *)
(* Institute: Colorado State University *)
(* Author: Mahmoud Abdelgawad *)
(* Date: Spring 2025 *)

(* check vehicle feul, LOS, and BLOS *)
fun vehicle_status(SoS:SoS_Formation) =
    let
        val Vehicle = #Vehicle SoS
        val feul = #feul Vehicle
        val LOS = #LOS Vehicle
        val BLOS = #BLOS Vehicle
    in
    if  (feul ≥ 70) andalso LOS andalso BLOS
     then
        1`Vehicle
    else
        empty
end;
(* ———————————————————————————————————————————— *)


(* check done battery, camera, IR sensor, and LOS *)
fun drones_status(SoS:SoS_Formation) =
    let
        val Drone = #Drone SoS
        val battery = #battery Drone
        val camera = #camera Drone
        val IR_sensor = #IR_sensor Drone
        val LOS = #LOS Drone
    in
    if  (battery ≥ 90) andalso camera
        andalso IR_sensor andalso LOS
     then
        1`Drone
    else
        empty
end;
(* ———————————————————————————————————————————— *)


(* deploy the mission-critical system formation *)
fun deploy_SoS(vehicle:VEHICLE, drones:DRONE) =
    let
        (* Vehicle data *)
        val vehicle_location = #location vehicle
```

```
        val vehicle_GPS = #GPS vehicle
        val vehicle_speed_read = #speed vehicle
        val vehicle_speed = (vehicle_speed_read + 10)
        val vehicle_feul = #feul vehicle
        val vehicle_LOS = #LOS vehicle
        val vehicle_BLOS = #BLOS vehicle
        val evaluation_reported = #evaluation_reported vehicle
        (* Drones data *)
        val drone_location = #location drones
        val drone_GPS = #GPS drones
        val drone_fly = #fly drones
        val drone_accel_read = #accel drones
        val drone_accel = (drone_accel_read + 10)
        val drone_battery_read = #battery drones
        val drone_battery = (drone_battery_read - 1)
        val drone_camera = #camera drones
        val drone_IR_sensor = #IR_sensor drones
        val drone_LOS = #LOS drones
        val data_collected = #data_collected drones
    in
    1`{Vehicle= {location = vehicle_location, GPS = vehicle_GPS,
     speed = vehicle_speed, feul = vehicle_feul, LOS = vehicle_LOS,
        BLOS =  vehicle_BLOS, evaluation_reported = evaluation_reported},
        Drone = {location = drone_location, GPS = drone_GPS,
        fly = drone_fly, accel = drone_accel, battery = drone_battery,
        camera = drone_camera, IR_sensor =  drone_IR_sensor,
        LOS = drone_LOS, data_collected= data_collected}}

    end;
(* ------------------------------------------------- *)


(* get the vehicle object from the formation *)
fun get_vehicle(SoS:SoS_Formation) =
    let
        val Vehicle = #Vehicle SoS
    in
        1`Vehicle
end;
(* ------------------------------------------------- *)


(* get the dones objects from the formation *)
fun get_drones(SoS:SoS_Formation) =
    let
        val Drone = #Drone SoS
    in
        1`Drone
```

```
end;
(* ———————————————————————————————————————— *)


(* check the battery level *)
fun battery_high(drone:DRONE) =
    let
        val battery = #battery drone
    in
    if battery ≥ 2
     then
        1`drone
    else
        empty
end;
(* ———————————————————————————————————————— *)


(* drones scanning the area *)
fun scan(drones:DRONE) =
    let
        val location = #location drones
        val GPS = #GPS drones
        val fly = #fly drones
        val accel = #accel drones
        val battery = #battery drones
        val camera = #camera drones
        val IR_sensor = #IR_sensor drones
        val LOS = #LOS drones
        val data_collected = #data_collected drones
    in
    if camera = true
    then
        2`{location = location, GPS = GPS, fly = fly, accel = accel,
            battery = battery, camera = camera, IR_sensor =  IR_sensor,
            LOS = LOS, data_collected= true}
    else
        2`{location = location, GPS = GPS, fly = fly, accel = accel,
            battery = battery, camera = camera, IR_sensor =  IR_sensor,
            LOS = LOS, data_collected= false}
end;
(* ———————————————————————————————————————— *)


(* update battery one unite - subroutine *)
fun update_battery_one_unit(x) =
    if x ≥1
        then
```

```
            x-1
        else
            x;
(* ———————————————————————————————————————— *)


(* update battery - subroutine *)
fun update_battery(SoS:SoS_Formation) =
    let
        val drones = #Drone SoS
        val location = #location drones
        val GPS = #GPS drones
        val fly = #fly drones
        val accel = #accel drones
        val battery_read = #battery drones
        val battery = if battery_read > 1 then
            update_battery_one_unit(battery_read) else battery_read
        val camera = #camera drones
        val IR_sensor = #IR_sensor drones
        val LOS = #LOS drones
        val data_collected = #data_collected drones
    in
        1`{location = location, GPS = GPS, fly = fly, accel = accel,
            battery = battery, camera = camera, IR_sensor =  IR_sensor,
            LOS = LOS, data_collected= data_collected}
end;
(* ———————————————————————————————————————— *)



(* reformation the critical-mission system *)
fun reform(vehicle:VEHICLE, drones:DRONE) =
    1`{Vehicle= vehicle, Drone = drones};
(* ———————————————————————————————————————— *)


(* check the vehicle BLOS *)
fun BLOS_On(Vehicle:VEHICLE) =
    let
        val BLOS = #BLOS Vehicle
    in
    if  BLOS = true
     then
        1`Vehicle
    else
        empty
end;
(* ———————————————————————————————————————— *)
```

```
191
192
193  (* check the vehicle LOS *)
194  fun LOS_On(Vehicle:VEHICLE) =
195      let
196          val LOS = #LOS Vehicle
197      in
198      if  LOS = true
199       then
200          1`Vehicle
201      else
202          empty
203  end;
204  (* ------------------------------------------------- *)
205
206  (* check the mission status *)
207  fun status(SoS:SoS_Formation) =
208      let
209          val drones = #Drone SoS
210          val drones_location = #location drones
211          val data_collected = #data_collected drones
212          val Vehicle = #Vehicle SoS
213          val vehicle_location = #location Vehicle
214          val evaluation_reported = #evaluation_reported Vehicle
215
216      in
217      if  drones_location = vehicle_location andalso
218          data_collected  = true andalso evaluation_reported = true
219       then
220          1`SUCCEED
221      else if drones_location = vehicle_location andalso
222          data_collected  = false orelse evaluation_reported = false
223          then
224          1`INCOMPLETE
225      else
226          1`FAIL
227  end;
228  (* ------------------------------------------------- *)
229
230
231  (* instruct to engage *)
232  fun is_instructed(SoS:SoS_Formation) =
233      1`SoS;
234  (* ------------------------------------------------- *)
235
236
237  (* update the vehicle location *)
238  fun update_vehicle_location(SoS:SoS_Formation) =
```

```
    let
        val vehicle = #Vehicle SoS
        val location_read = #location vehicle
        val vehicle_location = location_read
        val vehicle_GPS = #GPS vehicle
        val vehicle_speed = #speed vehicle
        val vehicle_feul = #feul vehicle
        val vehicle_LOS = #LOS vehicle
        val vehicle_BLOS = #BLOS vehicle
        val evaluation_reported = #evaluation_reported vehicle
    in
    1`{location = vehicle_location, GPS = vehicle_GPS, speed = vehicle_speed,
        feul = vehicle_feul, LOS = vehicle_LOS, BLOS =  vehicle_BLOS,
        evaluation_reported = evaluation_reported}
end;
(* ———————————————————————————————————————— *)


(* update the dones location *)
fun update_drones_location(SoS:SoS_Formation) =
    let
        val drones = #Drone SoS
        val location_read = #location drones
        val drone_location = location_read
        val drone_GPS = #GPS drones
        val drone_fly = #fly drones
        val drone_accel = #accel drones
        val drone_battery = #battery drones
        val drone_camera = #camera drones
        val drone_IR_sensor = #IR_sensor drones
        val drone_LOS = #LOS drones
        val data_collected = #data_collected drones
    in
        1`{location = location, GPS = GPS, fly = fly, accel = accel,
            battery = battery, camera = camera, IR_sensor =  IR_sensor,
            LOS = LOS, data_collected= data_collected}
end;

(* ———————— end ———————————————————— *)

```