Workflow Resilience For Mission Critical Systems

Mahmoud Abdelgawad, Indrakshi Ray, and Tomas Vasquez

Department of Computer Science, Colorado State University
Fort Collins Colorado 80523
{M.Abdelgawad,Indrakshi.Ray,Tomas.Vasquez}@colostate.edu

Abstract. Mission-critical systems, such as navigational spacecraft and drone surveillance systems, play a crucial role in a nation's infrastructure. Since these systems are prone to attacks, we must design resilient systems that can withstand attacks. Thus, we need to specify, analyze, and understand where such attacks are possible and how to mitigate them while a mission-critical system is being designed. This paper specifies the mission-critical system as a workflow consisting of atomic tasks connected using various operators. Real-world workflows can be large and complex. Towards this end, we propose using Coloured Petri Nets (CPN), which has tool support for automated analysis. We use a drone surveillance mission example to illustrate our approach. Such an automated approach is practical for verifying and analyzing the resiliency of mission-critical systems.

Keywords: Mission-critical Systems · Workflow · Coloured Petri Nets.

1 Introduction

A mission-critical system is one whose failure significantly impacts the mission [9][17]. Examples of mission-critical systems include navigational systems for a spacecraft and drone surveillance systems for military purposes.[7] These systems are prone to attacks because they can cripple a nation [6]. Mission-critical systems must fulfill survivability requirements so that a mission continues in the face of attacks. Thus, this requires specifying and analyzing a mission before deployment to assess its resilience and gauge what failures can be tolerated.

A mission can be described in the form of a workflow consisting of various tasks connected via different types of control-flow operators. Researchers have addressed workflow resiliency in the context of assigning users to tasks [8,14,16,20,22,13,15]. However, active attackers can compromise the capabilities of various entities. The destruction of the capabilities may cause the mission to abort or fulfill only a subset of its objectives. Analyzing resiliency considering attacker actions for mission-critical systems is yet to be explored.

Our work aims to fill this gap. We formally specify a mission in the form of a workflow, the definition of which is adapted from an earlier work [21]. A mission is often complex, and manually analyzing the workflow is tedious and error-prone. We demonstrate how such a workflow can be transformed into a Coloured Petri Net (CPN). CPN has automated tool support [18] that can be used for formal analysis.

Formal analysis may reveal deficiencies in the mission specification. Addressing such deficiencies improves the cyber-resiliency posture of the mission. We demonstrate

our approach using a mission-critical drone surveillance system. We provide a divide-and-conquer approach that helps decompose a complex workflow and shows how to analyze the sub-parts and obtain the analysis results for the total workflow.

The rest of the paper is organized as follows. Section 2 provides the formal definitions for workflow and Coloured Petri Nets. Section 3 describes a motivating example and formally represents the workflow for mission-critical systems. Section 4 defines the transformation rules from workflow to CPN along with the development of the CPN hierarchy model. Section 5 focuses on verifying the CPN so-generated and analyzing resiliency. Section 6 enumerates some related work. Section 7 concludes the paper and points to future directions.

2 Background

2.1 Workflow Definition

Typically, a workflow consists of tasks connected through operators [1,2,3,4,5,21]. The syntax of the workflow adapted from [21] 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 a unique initial task and a 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 valued 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 the 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 a simple or a compound one.*

Definition 4 (Entities). *The tasks of a workflow are executed by active entities, referred to as* subjects*. The entities on which we perform tasks are referred to as* objects*. An entity has a set of typed variables that represents 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 some task can be performed on it.*

In the remainder of this paper 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 *is the set initial conditions, and* O *is the set of mission objectives. The subject to task assignment should satisfy the access control policies of the mission. The conditions and objectives are expressed in predicate logic.*

2.2 Coloured Petri Nets (CPN)

A Colored Petri Net (CPN) is a directed bipartite graph, where nodes correspond to 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 of 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 by using the label r for transitions. CPNs operate on multisets of typed objects called tokens. 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 [10][11] 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, 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 [11] 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 place* i' *and a single output port place* o'.

Definition 10. Hierarchical CPN *A* Hierarchical CPN *is defined by the authors in [11] as:* $CPN_H = (S, SM, PS, FS)$ *where (i) S is a finite set of modules. Places and transitions must be disjoint from all other modules' places and transitions. For a module* $s \in S$ *we use the notation* P^s *to denote the set of places of the module s. Similarly, for each of the other elements of the module s. (ii)* $SM \subseteq T_{sub} \times S$ *is a relation that maps each substitution transition to a sub- module. Note that [10,11] defines* $SM : T_{sub} \rightarrow S$ *as a function. However, it is easier to compute SM if defined as a set. (iii)* $PS(t) \subseteq P_{sock}(t) \times P_{port}^{SM(t)}$ *is a port-socket relation function that assigns a port-socket relation to each substitution transition. (iv)* $FS \subseteq 2^p$ *is a set of non-empty fusion sets. A fusion set is a set of places that are functionally equivalent. (v) We additionally define global sets of places, transitions, arcs, colors, and variables as the union of the places, transitions, arcs, colors, and variables of each module. Furthermore, we define global initialization, arc expression, guard expression functions to be consistent with the functions defined for each module. We refer to these global elements by omitting the superscript in the notation.*

Definition 11 (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 the module* s_2*. As represented in [11], 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.

3 Motivating Example

We shall refer to the surveillance drone mission example to illustrate the transformation framework. Let $\mathcal{M}_{drone}$ be a mission specification that models a drone performing some data collection tasks over a region of interest. The drone has a camera that can take pictures at high and low altitudes and sensors capturing heat signals and radiation levels. The sensors are only accurate at low altitudes. There are three regions of interest: Regions A, B, and C. Region A is where the drone is regularly scheduled to perform surveillance; this region is large but close to the deployment point. Regions B and C are smaller but are further away from the deployment point. The drone is likely to be detected at low altitudes when it is in Region A. Therefore, the drone can only fly at high altitudes from where it can only use its camera. The drone may fly over regions B or C at high or low altitudes. However, due to the lack of visibility over the regions, only the heat and radiation sensors can capture meaningful data. Therefore, the drone can only collect data with its sensors over regions B and C. The mission succeeds if the drone collects data and returns to the deployment point. The control-flow of this mission is described by the task graph in Figure 1 and given as control-flow expression below:

$$W = Init \otimes Check\_Status \otimes Deploy \otimes if(instruction)\{(Fly\_to\_Region_B \,\#\, Fly\_to\_Region_C) \otimes (Measure\_Radiation\_level \,\&\, Measure\_Heat\_Signal)\} else\{Fly\_to\_RegionA \otimes while\{battery\_level > 1\}\{Scan\_Vehicles \,\&\, Scan\_Construction\}\} \otimes Return\_to\_Base \otimes Final$$

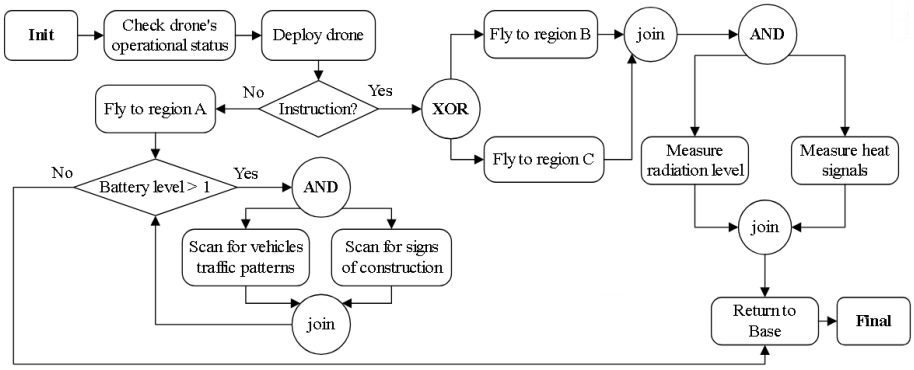

Fig. 1. Workflow of Surveillance Drone Mission.

This mission has a single entity called $drone_1$ of type *Drone*, that is, $\mathcal{S} = \{drone_1 : Drone\}$. The attributes of type *Drone*, denoted as *Drone.Attributes*, are given as:
$Drone.Attributes = \{type : string; location : string; fly\_enabled : bool; battery\_level \in \{1,2,3,4\}; instruction\_issued : bool; camera\_enabled : bool; sensors\_enabled : bool; data\_collected : bool\}$

We define functions that return the values of various attributes. For example, $location(drone_1)$ returns the location of $drone_1$. Every task can be performed by $drone_1$. Therefore, the subject task assignment function assigns $drone_1$ to each task. $\mathcal{ST} = \{(drone_1, transition) | t \in Tasks(W)\}$. Let I be a predicate logic formula giving the initialization conditions as:
$I = \exists s \in \mathcal{S} \,|\, (type(s) = Drone) \wedge (location(s) = \text{“}base\text{”})$
$\wedge (fly\_enabled(s) = true) \wedge (battery\_level(s) = 4) \wedge (instruction\_issued(s) = false) \wedge$
$(camera\_enabled(s) = true) \wedge (sensors\_enabled(s) = true) \wedge (data\_collected = false)$
Let O be a predicate logic formula that defines the mission objectives as:
$O = \exists s \in \mathcal{S} \,|\, (type(s) = Drone) \wedge (location(s) = \text{“}base\text{”}) \wedge (data\_collected(s) = true)$
If an attribute of the subject s is not explicitly constrained by the initial conditions or the objective, then that attribute can take on any value in its domain. Mission specification is as follows: $\mathcal{M}_{drone} = (W, \mathcal{S}, \mathcal{ST}, I, O)$.

4 Workflow to CPN Transformation Rules

Our approach maps a Mission-Specification $\mathcal{M} = (W, \mathcal{S}, \mathcal{ST}, I, O)$ to a Hierarchical Colored Petri Net $CPN_H = (S, SM, PS, FS)$. The approach follows six processes as shown in Figure 2. The first two processes deal with the decomposition and simplification of the control-flow. The decomposition procedure partitions the control-flow into a set of disjoint expressions that each model a workflow component. Each component is either a simple or compound workflow. The simplification procedure then iterates each component substituting any nested components with tasks. A task substituting a nested workflow in another workflow is *substitution task* . A workflow that has had all of its components substituted for tasks is said to be *simplified*. The simplification

process outputs a set of simple workflows, a set of substitution tasks for each component, and a substitution relation that tracks which component was substituted by which task. The formalization process extracts from the simple workflow the sets of workflow tasks, substitution tasks, begin and end tasks, and precedence constraints over all tasks. These sets are collectively called a *formalized component*.

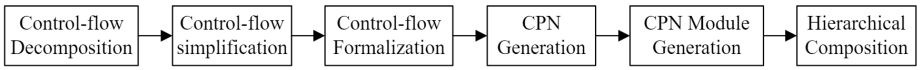

Fig. 2. Workflow to CPN Transformation Framework

The fourth process applies 7 rules to each component that, together with the remaining elements of the mission, are mapped to a Non-Hierarchical Colored Petri Net. The structure of each Non-Hierarchical CPN (places, transitions, and arcs) is semantically equivalent to the component. The fifth process generates a CPN module by adding an interface to each Non-Hierarchical CPN. The sixth process relates each module through relationships between substitution transitions, sub-modules, and port and socket places. The final output is a Hierarchical CPN representing a complete model of the original mission. Table 1 illustrates the notation used to decompose,simplify and formalize the control-flow. Note that the sets $Task(W)$,$Task_S(W)$, and $Task_U(W)$ are disjoint subsets of the set of all tasks $Tasks_W$. i.e. $Task(W) \cap Task_S(W) \cap Task_U(W) = \emptyset$

Symbol	Description
$Tasks_W$	Set of all tasks in a workflow
$Tasks(W) \subseteq Tasks_W$	Set of all workflow tasks in W
$Components$	Set of component workflows
$Tasks_S(W) \subseteq Tasks_W$	Set of substitution tasks of W
$Tasks_U(W) \subseteq Tasks_W$	Set of support tasks of W
$Tasks_B(W) \subseteq Tasks_W$	Set of begin tasks of W
$Tasks_E(W) \subseteq Tasks_W$	Set of end tasks of W
$Prec(W)$	The set of precedence constraints in W
$Substitutions$	Relation between component workflows and substitution tasks of W.

Table 1: Control-flow Notation Table

Transformation Rules The structure of the Non-Hierarchical CPN is made up of the places P', transitions T', and arcs A'. We define the structure of Non-Hierarchically CPNs in a similar manner as a workflow-net or process net [2,1,19]. A workflow-net is a Petri Net with a unique input place i and a unique output place o. A workflow-net has the additional property that when the output place o is connected to the input place i by a transition r', the resulting *extended net* is strongly connected.

Rule 1: Transition set generation: Tasks are modeled as transitions in the Non-Hierarchical CPN. Our algorithm maps each task $t_k \in Tasks_{W'}$ to a unique transition $r_k \in T'$. Let T'_{map} be a relation between the set of tasks and the set of transitions. The pair $(t_k, r_k) \in T'_{map}$ indicates that the transition r_k is mapped from the task t_k. Through the remainder of the paper, we maintain this indexing convention. That is, r_k denotes the transition mapped from the task t_k. Let T'_{sub}, T'_U, T'_B and T'_E be subsets of T'. The set T'_{sub} is the set of substitution transitions such that $T'_{sub} = \{r_s \,|\, (t_s, r_s) \in T'_{map} \wedge t_s \in Tasks_S(W')\}$. The sets of support transitions T'_U, begin transitions T'_B, and end transitions T'_E are constructed similarly from the respective subsets of tasks. Consider our running example, the sets of transitions for CPN' are mapped from the sets of tasks of W' as follows: $Tasks'_W \rightarrow T' = \{r_1, r_{s1}, r_{s5}, r_5, r_c\}$, $Tasks_U(W') \rightarrow T'_U = \{r_c\}$, $Tasks_S(W') \rightarrow T'_S = \{r_{s1}, r_{s5}\}$, $Tasks_B(W') \rightarrow T'_B = \{r_1\}$, and $Tasks_E(W') \rightarrow T'_E = \{r_5\}$. The relation between tasks and transitions is $T'_{map} = \{(t_1, r_1), (t_{s1}, r_{s1}), (t_{s5} r_{s5}), (t_5, r_5), (t_c, r_c)\}$.

Rule 2: Place and arc set generation: The set of places P' is initialized with a unique input place i and a unique output place o. The set of arcs is initialized to the empty set. For each begin transition $r_b \in T'_B$ the algorithm adds a directed arc connecting the input place i and the transition r_b to A', i.e. $A' \leftarrow (i, r_b)$. Similarly, for each end transition $r_e \in T'_E$, the algorithm adds a directed arc connecting the the transition r_b and the input place i to A', i.e. $A' \leftarrow (r_e, o)$. Consider our running example, $r_1 \in T'_B$ implies that $A' \leftarrow (i, r_1)$ and $r_5 \in T'_E$ implies $A' \leftarrow (r_5, o)$. For each (t_k, t_j) in $Prec(W')$, we create a new place m and add the arcs (r_k, m) and (m, r_j) to A'. That is, $P' \leftarrow m$, and $A' \leftarrow (t_k, m), (m, t_j)$. If one task has two direct successors i.e. $(t_q, t_j), (t_q, t_k) \in Prec(W')$, then this denotes a point at which a split occurs. If a task has two direct predecessors, i.e. $(t_j, t_q), (t_k, t_q) \in Prec(W')$, then this denotes a point at which a join occurs. There are two types of splits and joins. There are or-splits/joins and there are and-splits/joins [19]. Or-splits should always be joined by an or-join. Similarly, an and-split should always be joined by an and-join. The or-split/join only occurs in control-flows that contain the exclusive-or operator or the conditioning operator. When an or-split is imposed by the exclusive-or operator and there is task t_q that directly precedes $t_j \# t_k$, such as $t_q \otimes (t_j \# t_k)$, the transition r_q should output to a single place m that is the input to both r_j and r_k. Then the either transition r_j or r_k will execute by consuming the output of r_q, but not both. We thus leverage the fact that given a place m that is input to two transitions, then either transition may consume the token in the place m. When an or-split is imposed by the conditioning operator, then the split is modeled by the support transition r_c- where r_c routs the execution based on the evaluation of a Boolean expression. Therefore, $(t_q, t_j), (t_q, t_k) \in A'$ and an exclusive-or operator in the control-flow implies that $P' \leftarrow m$, $P' \leftarrow m'$, and $A' \leftarrow (r_q, m), (r_q, m'), (m, r_j), (m', r_k))$, however, $m = m'$. Similarly, when an or-join is imposed by the exclusive-or operator or the conditioning operator, and there is task t_q that directly succeeds $t_j \# t_k$, such as $(t_j \# t_k) \otimes t_q$, then the output place of r_j and r_k should be a single place that is the input to r_q. Therefore, $(t_j, t_q), (t_k, t_q) \in A'$ and the conditioning operator or exclusive-or is in the control-flow implies that $P' \leftarrow m$ and $P' \leftarrow m'$, and $A' \leftarrow (r_j, m), (r_k, m'), (m, r_q), (m', r_1)$, however, $m = m'$. The and-split/join only occurs in control-flows that contain the parallel-split operator. The and-split is modeled by the support transition r_g. The and-join is modeled by the support transition r_s. The structure of CPN' is formally described as:

$P' = \{i, o, p_1, p_2, p_3, p_4\}$
$T' = \{r_1, r_{s1}, r_{s5}, r_5, r_c\}$
$A' = \{(i, r_1), (r_1, p_1), (p_1, r_c), (r_c, p_2), (r_c, p_3), (p_2, r_{s1}), (p_3, r_{s5}), (r_{s1}, p_4), (r_{s5}, p_4), (p_4, r_5), (r_5, o)\}$

Thus far we have addressed five of the six well-behaved building blocks proposed by the Workflow Management Coalition [19] to model any control-flow. That is, the and-split, and-join, or-split, or-join, and sequence. The final building block is iteration. In the special case that the simple workflow contains the iteration operator, our algorithm adds two additional arcs to the set of arcs. One directed arc going from the support transition r_{c1} to the output place o models the iteration never executing. One directed arc going from the support transition r_{c2} to the output place of r_{c1} models the iteration continuing. One can see that our model can implement all six well-behaved building blocks proposed by the Workflow Management Coalition [19] and can therefore model any control-flow. Furthermore, we will later show that the manner in which we connect our CPN modules forms well behaved control structures [19].

Rule 3: Colors and variable set generation: The color set of a simple workflow CPN consists of the different types of entities in the Simple CPN model. The types can be primitive types such as *Bool*, *Int*, or *String* or more complex user-defined types. In the drone surveillance example, there is only one subject of type *Drone*. The data type *Drone* is mapped to a record color set labeled *Drone*. Each attribute of the type *Drone* becomes a label in the record color set as:

$$color\,Drone = \textbf{record}\ \{type : String; location : String; fly\_enabled : Bool; \\ detected : bool; battery\_level : Int \in \{1,2,3,4\}; instruction\_issued : Bool; \\ sensors\_enabled : Bool; camera\_enabled : Bool; data\_collected : Bool\}$$

Set of colors Σ' is constructed by adding any compound color sets and declaring the primitives that compose them. $\Sigma' = \{Bool, String, Int, Drone\}$ Set of variables V is declared such that there is variable for each color in Σ. $V' = \{instruction : Bool; y : String, i : Int, drone : Drone\}$.

Rule 4: Assigning colors to places: Each task can be performed by a drone. Therefore, each place $p \in P'$ is assigned the color $Drone \in \Sigma'$.

Rule 5: Assigning guard expressions: For each workflow task $t_k \in Tasks_{W'}$, we assign to the transition r_k a guard expression $G'(r_k)$ equivalent to $Pre(t_k)$. In other words, the guard expression evaluates to true if and only if the conjunction of the preconditions of t_k evaluates to true. The empty set of pre conditions is equivalent to the pre condition that always evaluates to true. Support and substitution tasks always have an empty set of pre conditions, therefore the guards of the respective transitions always evaluate to true. The only tasks of W' that have non-empty sets of pre conditions are t_1 and t_5 such that $Pre(t_1) = \{fly\_enabled(drone_1) = true\}$ and $Pre(t_5) = \{fly\_enabled(drone_1) = true, battery\_level(drone_1) \geq 1\}$. Therefore, the guard expression function is $G'(r_1) = g_1 = fly\_enabled(drone) = true$ and $G'(r_5) = g_5 = fly\_enabled(drone) = true \wedge$
$battery\_level(drone) \geq 1$. Where $drone \in V'$ takes on the value of the instance of $drone_1$ when r_1 and r_2 are enabled. For all other cases in *CPN*‘ the guard expression is always true.

Rule 6: Assigning arc expressions: Each transition is assigned an input arc expression that evaluates to a token of the same color as the input place of the transition. Each

transition is assigned an output arc expression that updates the token received over the input arc, such that the post conditions of the corresponding task evaluate to true. Substitution transitions can neither be enabled nor occur. Therefore, the arc expressions over their connected arcs have no semantic meaning. Support transitions route the execution of the workflow. For sequential and parallel executions, the output arc expressions are identical to the input arc expressions. For support transitions that evaluate a condition, each output arc models a case of the condition. The output arc of these transitions, should output the token received over the input arc only if the case evaluates to true. Otherwise, they output the empty set.

For example, for the arc $(r_c, p_2) \in A'$, is given the arc expression $E'((r_c, p_1)) = c_1 = if(instruction\_issued(drone) = true)\{drone\}else\{empty\}$. The arc (r_5, o) is given the arc expression $E'((r_5, o)) = e_5 = \{(location(drone) = \text{“}deployment\_point\text{”}) \wedge (battery\_level(drone) = battery\_level(drone) - 1)\}$. Note that the syntax we use for the arc expressions is based on function notation that is easy to understand. CPN Tools has its own modeling language which we avoid using for simplicity.

Rule 7: Initialization: The initialization function I' sets the initial state of the model by assigning a multiset of tokens to each place $p \in P'$. I must satisfy the initial conditions of the mission. Recall, I calls for a subject s with the following valuation: $(type(s) = Drone) \wedge (location(s) = \text{“}deployment\_point\text{”}) \wedge (fly\_enabled(s) = true) \wedge (battery\_level(s) = 4) \wedge (instruction\_issued(s) = false)) \wedge (camera\_enabled(s) = true) \wedge (sensors\_enabled(s) = true) \wedge (data\_collected = false)$

Over the input place i, the initialization function $I'(i)$ evaluates to a multiset of size one that satisfies the initial conditions of the workflow. For every other place, the initialization function evaluates to the empty set of tokens. It is important to note that for our analysis we are considering a single drone in isolation. Therefore, the initialization function of every other CPN in our final hierarchical model will evaluate to the empty set for every place. The result of applying rules 1-7 to our example W' is CPN' as described by Figure 3. We have omitted writing each arc expression explicitly into the diagram to maintain readability.

We construct a hierarchical CPN (CPN_H) from the set of modules S and their original relationships in the workflow. Recall that a Hierarchical Colored Petri Net $CPN_H = (S, SM, PS, FS)$ consists of a set of modules S, an assignment of substitution transitions to modules SM, a relation between port places and socket places PS -where the pair $(p, p') \in PS$ signifies that p is a socket place of a substitution transition t that has been mapped to a a sub module s such that p' is a port place of the same type as p; and a set of fusion places FS. We have already defined and computed the set of modules S. The set of fusion places is empty e.g. $FS = \emptyset$. Therefore, the task is now to compute SM and PS. The module CPN'_M is the prime module (top level of the hierarchy) and every other module is a sub-module with respect to CPN'. The initial marking M_0 always evaluates to a non-empty set of tokens at the the input port $i' \in P'$; and every other place is empty. The place i' has a single output arc connected to the transition r_i. Thus, all execution sequences must begin at r_i. The model has a single final transition r_f with a single output place o'. Therefore, any execution sequence that is complete must terminate with the transition r_f and a token in the place o'. The initial transition $r_i \in T'$ and final transition $r_f \in T'$ model the begin and end tasks of the original workflow.

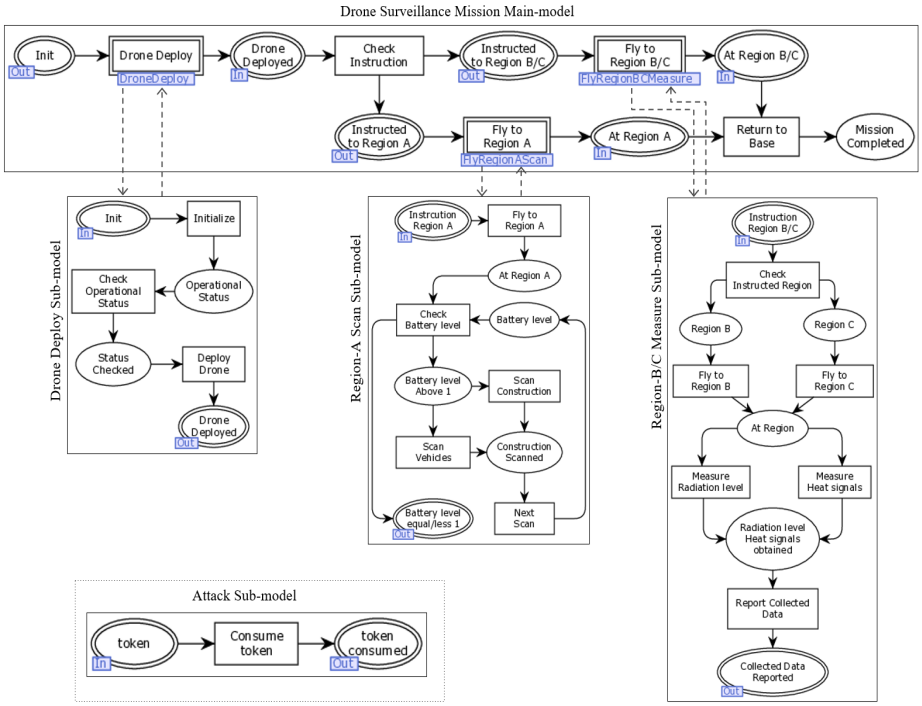

Fig. 3. CPN Models of Surveillance Drone Mission.

5 Resiliency Analysis

An attack is an action taken by an adversary that changes the state of a subject in a way that renders it unable to perform a task it has been assigned. An attack scenario consists of the target of the attack. The set of attributes of the target, which are attackable, an attack task, and an integer limit on the number of times the attack may occur. A mission workflow is resilient to an attack scenario if, after the attack occurs, there exists a successful execution from the point at which the attack occurred.

There are various scenarios of workflow resilience, including static, decremental, and dynamic resilience [20] (see section 6). In this paper, we investigate static resilience. Static resilience describes a situation where a subset of users become unavailable before the execution of the workflow. Once a subset of users is made unavailable, no user of this subset may become available again.

The analysis examines the state space of the CPN model with the attack scenario where the drone's camera fails while scanning region A since we know that it is the only location where the drone uses its camera. Table 2 reports 57 nodes and 65 arcs that are strongly connected components (SCC). It also reports that model transitions are fully executed. However, the state space report has two dead markings (42 and 57). The first dead marking (42) has a drone in the output place and received instruction. The second

dead marking (57) has a drone in the output place and did not receive an instruction. Therefore, the mission is guaranteed to terminate in success under an attack scenario.

State Space		**SCC Graph**		**Status**
#Nodes	#Arcs	#Nodes	#Arcs	Full
57	65	57	65	
Dead Markings		Dead Transition		Live Transition
[42,57]		None		None

Table 2: State Space Verification

We then write a program using the CPN-ML programming language [12] to analyze the mission's resilience. We then use the CPN State Space Analysis Tool to evaluate the program functions and return an execution sequence for each outcome where the attack succeeded. The program then backtracks through the shortest execution sequence and finds the node in the state space representing the state where the attack occurred. From that state, it searches for an execution sequence that results in a successful outcome (attack failure). If it finds one, it returns the execution sequence. Otherwise, it returns an empty list. The program also returns information about the set of dead markings and partitions the set into three subsets. The dead markings represent outcomes where the attack did not occur, the attack occurred, and the mission failed, and where the attack occurred, and the mission succeeded.

Table 3 summarizes the resiliency analysis result. The first column describes the scanning iteration of Region A. The first row shows 37 total dead markings (DM), 2 dead markings that represent outcomes where the attack did not occur (DNA), 10 dead markings that represent outcomes where the attack occurred but failed (DAF), 25 dead markings that represent outcomes where the attack occurred and succeeded (DAS). The algorithm found paths that reach only 4 of the 25 DAS markings (RDAS).

Scan Iteration Number	Dead Markings (DM)	Attack Didn't Occur (DNA)	Attack Occurred but Failed (DAF)	Attack Occurred and Succeed (DAS)	Reachable DAS (RDAS)
1	37	2	10	25	4
2	33	2	10	21	0
3	16	2	9	5	0
4	12	2	10	0	0

Table 3: Static Resilience Analysis Results

The program inspects the set of DNA markings and returns that the mission is correct and valid. The DNA markings represent the original outcomes of the mission. The size of this set should be the same as the original set of dead markings. For the 25 failed outcomes, the program found a successful execution sequence for four outcomes.

We compare the successful execution sequence with the failed execution sequence for each of these four outcomes. It leads us to find the state where the execution sequences diverge and, from that state, ensure that the transition and binding element that leads to success always occur. For instance, node 42 corresponds to a failed outcome where the attack happened at node 3. From node 3, our program found a path to node 57 - a successful outcome. We now compare the path from node 3 to node 57 and the path from node 3 to node 42. We find that the execution sequences diverge at node 5. From node 5, proceeding to node 10 or node 9 is possible. The change in state from node 5 to node 10 results from the transition with the variable instruction bound to the value $false$. The change in state from node 5 to node 9 results from the transition with the variable instruction bound to the value $true$. We now know that the attack succeeds if the drone's battery is less than 3 units in the state represented by node 5, where transition $receiveinstruction$ is enabled. Transition receive instruction models the workflow routing based on an instruction issued.

In Iteration 2, the state space is re-calculated, and the analysis is repeated. The second row of table 3 describes the statistics generated by the program's second iteration. The result is four fewer outcomes where the attack succeeded. The program returns a set of nodes representing the states where the attack occurred, and no path to a successful outcome exists. Note that these are not dead markings. From each of these nodes, there is a path to a dead marking that represents the failure of the mission (success of the attack).

In Iteration 3, the state space is re-calculated, and the analysis is repeated. The third row of table 3 summarizes the statistics generated by the program's second iteration. We expected the set of outcomes where the attack failed to remain at 10, now 9. The reduced size of the failed attack set means the outcome is lost because the drone is initialized with its instruction set to $false$. Thus the attack at node 3, which was found to be resilient, never executes. We focus on the 5 markings where the attack occurred and succeeded. After inspecting all five markings, we find the attack succeeds because it is delivered when the drone has just enough battery to perform one additional pass over Region A. We can eliminate these failed outcomes by increasing the requirement on the loop over Region A from more than one unit of battery to more than two units of battery. It turns out that the drone always has some reserve battery if it is attacked.

In Iteration 4, the state space is re-calculated, and the analysis is repeated. The third row of table 3 summarizes the statistics generated by the program's second iteration. Since the set of attack successes is empty, we can be confident that the attack can not succeed under the restricted workflow.

In summary, we have shown how to assess a mission's resilience and find the conditions required for the mission to succeed. We have also shown how to restrict a workflow to improve its resilience.

6 Related Work

The literature on workflow resiliency problems introduces solutions to address unavailability [14,16,20]. Our work argues that the workflow resiliency problem can sometimes be viewed as unavailability and degradation. In other words, attacks do not per-

manently 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 since its other sensors can complete the workflow.

Wang *et al.* [20] introduce three types of resilience, static, decremental, and dynamic resilience. Static resilience refers to a situation in which users become unavailable before the workflow executes, and no users may become available during the execution. Decremental resiliency expresses a situation where users become unavailable before or during the execution of the workflow, and no previously unavailable users may become available during execution, while dynamic resilience describes the situation where a user may become unavailable at any time; a previously unavailable user may become available at any time. The different types of resilience formulations capture various types of attack scenarios.

Mace *et al.* [14][16] propose a quantitative measure of workflow resiliency. They use 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 the MDP models give a termination rate and an expected termination step.

7 Conclusion

This paper emphasizes the workflow resiliency of the task degradation problem, specifically for mission-critical cyber systems. We presented a set of rules that formally transforms workflow represented by a mission into Coloured Petri Nets (CPNs). We then solved various analysis problems related to the resiliency of mission-critical such as cyber-attacks. We developed an approach based on formalization rules that address the complexity of mission workflows, simplify them, and transform them into simple CPNs. These simple CPNs are then modulated and combined as a hierarchy CPN model.

We applied the approach to a drone surveillance system as an illustrative example. We used the CPN tools to run verification and reachability analysis. The results showed that the workflow resiliency problem could sometimes be unavailability and degradation. A workflow subject is not permanently removed from service when an attack occurs; it decreases its capabilities. However, the mission can continue, and the workflow can be completed. We have shown how to assess a mission's resilience and find the conditions to succeed. We have also shown how to restrict a workflow to improve its resilience.

Future work will focus on extending the generated model to account for multiple subjects and investigating decremental and dynamic resilience. We will design a set of algorithms corresponding to the transformation rules. Our end goal is to automate the process of verification and resilience analysis of workflows.

Acknowledgements This work was supported in part by funding from NSF under Award Numbers CNS 1715458, DMS 2123761, CNS 1822118, NIST, ARL, Statnett, AMI, NewPush, and Cyber Risk Research.

References

1. Van der Aalst, W.M.: Verification of workflow nets. In: International Conference on Application and Theory of Petri Nets. pp. 407–426. Springer (1997)
2. van der Aalst, W.: Structural characterizations of sound workflow nets (1996)
3. 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). https://doi.org/10.1023/A:1008758612291, https://doi.org/10.1023/A:1008758612291
4. Bride, H., Kouchnarenko, O., Peureux, F.: Verifying modal workflow specifications using constraint solving. In: Albert, E., Sekerinski, E. (eds.) Integrated Formal Methods. pp. 171–186. Springer International Publishing, Cham (2014)
5. Bride, H., Kouchnarenko, O., Peureux, F., Voiron, G.: Workflow nets verification: Smt or clp? In: ter Beek, M.H., Gnesi, S., Knapp, A. (eds.) Critical Systems: Formal Methods and Automated Verification. pp. 39–55. Springer International Publishing, Cham (2016)
6. Chong, J., Pal, P., Atigetchi, M., Rubel, P., Webber, F.: Survivability architecture of a mission critical system: The dpasa example. In: 21st Annual Computer Security Applications Conference (ACSAC'05). pp. 10–pp. IEEE (2005)
7. Flauzac, O., Mauhourat, F., Nolot, F.: A review of native container security for running applications. In: Shakshuki, E.M., Yasar, A. (eds.) The 17th International Conference on Mobile Systems and Pervasive Computing (MobiSPC 2020) / The 15th International Conference on Future Networks and Communications (FNC-2020) / The 10th International Conference on Sustainable Energy Information Technology, Leuven, Belgium, August 9-12, 2020. Procedia Computer Science, vol. 175, pp. 157–164. Elsevier (2020)
8. Fong, P.W.L.: Results in Workflow Resiliency: Complexity, New Formulation, and ASP Encoding, p. 185–196. Association for Computing Machinery, New York, NY, USA (2019)
9. 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)
10. Jensen, K., Kristensen, L., Wells, L.: Coloured petri nets and cpn tools for modelling and validation of concurrent systems. STTT **9**, 213–254 (05 2007). https://doi.org/10.1007/s10009-007-0038-x
11. Jensen, K., Kristensen, L.M.: Coloured Petri Nets: Modelling and Validation of Concurrent Systems. Springer Publishing Company, Incorporated, 1st edn. (2009)
12. Jensen, K., Kristensen, L.M.: CPN ML Programming, pp. 43–77. Springer Berlin Heidelberg, Berlin, Heidelberg (2009). https://doi.org/10.1007/b95112_3, https://doi.org/10.1007/b95112_3
13. Mace, J., Morisset, C., van Moorsel, A.: Modelling user availability in workflow resiliency analysis. In: Proceedings of the 2015 Symposium and Bootcamp on the science of security. pp. 1–10. HotSoS '15, ACM (2015)
14. Mace, J.C., Morisset, C., van Moorsel, A.: Quantitative workflow resiliency. In: Kutyłowski, M., Vaidya, J. (eds.) Computer Security - ESORICS 2014. pp. 344–361. Springer International Publishing, Cham (2014)
15. Mace, J.C., Morisset, C., van Moorsel, A.: Wrad: Tool support for workflow resiliency analysis and design. In: Software Engineering for Resilient Systems, pp. 79–87. Lecture Notes in Computer Science, Springer International Publishing, Cham (2016)

16. Mace, J.C., Morisset, C., Moorsel, A.v.: Impact of policy design on workflow resiliency computation time. In: Campos, J., Haverkort, B.R. (eds.) Quantitative Evaluation of Systems. pp. 244–259. Springer International Publishing, Cham (2015)
17. 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)
18. Rantzer, A.V.: CPN tools for editing, simulating and analysing coloured Petri nets. In: van der Aalst, W.M.P., Best, E. (eds.) Applications and Theory of Petri Nets 2003: 24th International Conference, ICATPN 2003. vol. 2679, pp. 450–462 (2003), http://http://cpntools.org
19. van der Aalst, W.M., ter Hofstede, A.H.: Verification of workflow task structures: A petri-net-baset approach. Information Systems **25**(1), 43–69 (2000). https://doi.org/https://doi.org/10.1016/S0306-4379(00)00008-9, https://www.sciencedirect.com/science/article/pii/S0306437900000089
20. Wang, Q., Li, N.: Satisfiability and resiliency in workflow authorization systems. ACM transactions on information and system security **13**(4), 1–35 (2010)
21. 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). https://doi.org/10.1109/TSC.2013.31
22. Zavatteri, M., Viganò, L.: Last man standing: Static, decremental and dynamic resiliency via controller synthesis. Journal of computer security **27**(3), 343–373 (2019)