Towards a Model-Driven Continuous Assurance Framework for Autonomous Systems

Dhaminda B. Abeywickrama[1[0000−0002−4423−0284]], Michael Fisher[1[0000−0002−0875−3862]], Frederic Wheeler[2[0009−0000−7760−4724]], and Louise Dennis[1[0000−0003−1426−1896]]

[1] Department of Computer Science, The University of Manchester, Manchester, UK
{dhaminda.abeywickrama, michael.fisher, louise.dennis}@manchester.ac.uk
[2] Regulatory Support Directorate, Amentum, Warrington, UK
frederic.wheeler@global.amentum.com

Abstract. Autonomous systems must sustain justified confidence in their correctness and safety throughout their operational lifecycle. Traditional assurance methods separate development-time assurance from runtime assurance, yielding fragmented arguments that cannot adapt to runtime changes or system updates. Towards addressing this, we propose a unified *Continuous Assurance Framework* integrating design-time, runtime, and evolution-time assurance within a traceable, model-driven workflow, and instantiate its design-time phase using two formal verification methods: RoboChart for functional correctness and PRISM for probabilistic risk analysis. We also propose a model-driven transformation pipeline, implemented as an Eclipse plugin, that automatically regenerates structured assurance arguments whenever formal *specifications* or their *verification* results change, ensuring traceability. We demonstrate our approach on a nuclear inspection robot scenario, and discuss alignment with regulator-endorsed best practices.

Keywords: Continuous assurance · Autonomous systems · Formal verification.

1 Introduction

Autonomous embodied agents operating in a physical environment typified by robotic autonomous systems (e.g., AIRs: autonomous inspection robots) must sustain justified confidence in their correctness and safety across design-time, deployment, and post-deployment evolution. Traditional assurance approaches often separate development-time and runtime evidence, leading to fragmented assurance arguments that are difficult to maintain as systems evolve. This is particularly significant for *assured autonomy*, where systems must sustain *trustworthiness* [2] while operating in dynamic and uncertain environments [20]. As autonomy increases, assurance arguments must *co-evolve* with the system.

Recent work highlights challenges in maintaining safety cases throughout the lifecycle of autonomous systems. Cârlan et al. [8] identify that existing maintenance processes are largely manual, costly, and error-prone; change impact

analyses can be overly pessimistic; quantitative assessment capabilities remain limited; and only narrow subsets of change scenarios are typically supported. Automation and traceability between assurance arguments and verification evidence remain only partially supported [8].

In our study, we distinguish between *static* and *dynamic* assurance [12, 21]. Static assurance refers to evidence and argumentation generated during development to justify a system's correctness prior to deployment. Dynamic assurance concerns operational confidence—i.e., whether the system continues to satisfy its safety requirements at runtime under changing conditions [12]. Additionally, we consider *evolution-time assurance* (during post-deployment), which addresses the preservation of assurance following system modifications (e.g., updates, feature extensions, or reconfiguration) [17]. We use *continuous assurance* to describe this holistic perspective: one that spans design-time, runtime, and evolution. This paper focuses on instantiating the design-time stage of the assurance process.

Formal verification (e.g., model checking) uses mathematical methods to establish whether a system satisfies a formal specification [13]. At design-time, we employ a heterogeneous approach combining two complementary model checking techniques: *RoboChart* with the FDR4 model checker to establish functional correctness and invariant preservation [16]; and *PRISM* for probabilistic model checking over Discrete-Time Markov Chains [18]. This combination supports both logical guarantees and quantitative assessment of key properties of an autonomous system. Meanwhile, *Model-Driven Engineering* (MDE) supports managing assurance artefacts and traceability as systems evolve [4, 21]. Building on this, we develop a *model-driven assurance pipeline*, implemented as an Eclipse plugin, that transforms PRISM artefacts (properties and verification results) into assurance arguments in Goal Structuring Notation (GSN) [19] and regenerates affected arguments when specifications or verification results change. An assurance case is a structured argument that provides justified confidence in claims about a system (e.g., safety) by combining explicit reasoning with supporting evidence, and is often expressed using notations such as GSN [9, 19].

This paper makes the following three contributions:

- We propose a conceptual framework for *continuous assurance* with three integrated phases—design-time, runtime, and evolution-time that define traceability mechanisms, assurance case arguments, and supporting automation.
- The framework's design-time phase is instantiated through two *formal verification* workflows—RoboChart for functional analysis, and PRISM for probabilistic analysis. Also, an MDE-based Eclipse plugin is proposed that transforms PRISM *specifications* and *verification* results into structured GSN assurance arguments, with automated regeneration when artefacts change.
- We explore our approach using an illustrative nuclear inspection scenario and discuss alignment with joint regulator-issued Trilateral AI Principles [7].

This paper is organised as follows. Section 2 provides key related work, and in Section 3, we describe our proposed continuous assurance framework. Section 4 presents the case study involving an illustrative nuclear radiation inspection scenario. Section 5 discusses the formal verification performed using RoboChart

and PRISM, and outlines the model-driven pipeline for generating GSN evidence. Section 6 discusses our approach, and Section 7 concludes with future directions.

2 Related Work

In this section, we review related work across design-time, runtime, and evolution-time assurance.

At the *design-time* stage, Bourbouh et al. [5] integrate formal verification results into safety arguments for an inspection robot via the AdvoCATE tool. However, their method primarily addresses static, design-time evidence and lacks mechanisms for runtime adaptation or automated evolution-time updates. Automation and traceability are also explored by Wei et al. [23], who embed validation rules directly into GSN models using Constrained Natural Language to enable executable traceability and consistency checks between assurance arguments and engineering artefacts.

For *runtime* assurance, Belle et al. [4] propose dynamic assurance cases for real-time safety assurance in autonomous driving, addressing aleatory uncertainty at design-time and managing epistemic uncertainty during operation through machine learning. Schleiss et al. [21] present a continuous safety assurance framework that uses runtime monitors to dynamically detect deviations from design-time assumptions and maintain assurance during operation. Dong et al. [11] investigate runtime safety case adaptation through automated argument modification in response to detected hazards. Collectively, these approaches highlight that static safety arguments risk obsolescence under evolving operational conditions of autonomous systems, but generally lack strong integration with design-time verification outputs.

At the *evolution-time* phase (post-deployment), methodologies such as ReASSURE [17] support structured change management by identifying assurance case elements impacted by updates or persistent runtime alerts and prioritising evidence regeneration using cost tags and compositional analysis. For self-adaptive and learning-enabled systems, Calinescu et al. [6] present the ENTRUST methodology, which combines design-time and runtime verification to support evolving assurance cases for systems whose configurations may change autonomously. Compared to ReASSURE, which focuses on structured reuse, impact analysis, and cost-aware evolution of assurance cases, our approach emphasises traceability and automated regeneration of assurance arguments directly from formal verification artefacts. Similarly, while ENTRUST integrates design-time and runtime evidence to support dynamic assurance cases, it does not provide automated transformation of probabilistic verification results into structured assurance arguments. Unlike prior work that addresses largely isolated lifecycle phases, we propose an interlinked process that supports monitoring, dynamic updates, and regeneration, enabling assurance cases to co-evolve with the autonomous system.

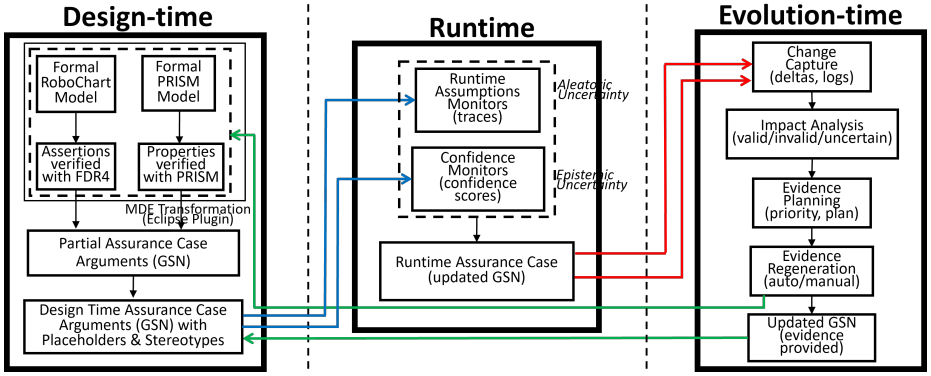

Fig. 1. Proposed continuous assurance framework with three stages.

3 A Model-Driven Continuous Assurance Framework

This section presents our unified *Continuous Assurance Framework* with three interlinked phases: *design-time*, *runtime*, and *evolution-time* (Fig. 1). Our framework targets the evolution of assurance cases rather than the internal structure of autonomous or AI components, whose behaviour is abstracted through formal models, assumptions, and evidence. It does not replace established lifecycle practices; instead, it focuses on maintaining traceability and regenerability of assurance case arguments across lifecycle phases. The MDE workflow in this paper consists of: (i) formal modelling and verification (RoboChart and PRISM), (ii) transformation of verification results into GSN assurance case arguments, and (iii) traceability mechanisms that enable future updates during runtime and evolution phases.

Design-time assurance (baseline and trace model) During the design-time phase, the system's behaviour and constraints are formally modelled and verified using RoboChart (with FDR4 model checker) and PRISM. Based on these analyses, a *partial assurance case* is created using GSN (Fig. 1), serving as the baseline for continuous assurance and automatically updated when the PRISM specification (properties) or verification results (evidence) change, while updates from RoboChart/FDR4 analysis are linked manually (Section 5.3).

In GSN, goals represent claims, strategies define argument structure, and solutions correspond to supporting evidence [19]. To support traceability and automated updates, we embed two types of structured annotations directly into GSN goal descriptions: *placeholders* and *stereotypes* (Table 1). Placeholders represent dynamic or deferred information, such as claims that must be validated at runtime or post-deployment (e.g., `trace_expr="..."`, `monitor_expr="..."`). For example, `trace_expr` captures environmental or control assumptions that guide runtime monitor configuration and violation detection. Stereotypes (e.g., `«TraceMonitored»`, `«DeferredEvidence»`) annotate GSN nodes with lifecycle-related information, supporting identification, filtering, and traceability across

the assurance case. The introduction of *placeholders* and *stereotypes* into the GSN model provides explicit, machine-interpretable annotations that can be used by analysis plugins and runtime monitoring components to identify claims that rely on deferred evidence, assumptions that are actively monitored, and argument nodes affected by changes in verification results or operational data. By embedding this metadata, the framework enables targeted evidence retrieval, selective re-verification, and focused runtime monitoring, avoiding exhaustive manual re-analysis. In practice, placeholders act as references that enable trace links once populated with concrete identifiers (e.g., references to PRISM models or properties), allowing changes in verification artefacts to be propagated to the corresponding GSN nodes. This trace infrastructure supports variability-aware workflows [17] and ensures that the assurance argument remains tightly coupled to the verified system model as requirements or configurations evolve.

Runtime assurance (dynamic monitoring and evidence update) Runtime assurance extends the design-time argument by introducing monitoring mechanisms that detect violations of assumptions during operation. At runtime, assumptions about sensing, control, and hazard response are treated as monitorable elements of the assurance case. During operation, the system may encounter both *aleatory* and *epistemic* uncertainty [21] that can invalidate design-time assumptions. To maintain assurance, we incorporate two types of runtime monitors into the GSN model—*runtime assumption monitors* and *confidence monitors*—each defined using structured placeholders and stereotypes. The former address violations due to aleatory uncertainty (random noise, data gaps), while the latter manage epistemic uncertainty (previously unknown risks revealed during operation) [21].

Runtime assumption monitors detect violations of design-time assumptions about sensor data or control-loop behaviour. Each monitor is represented by annotating a GSN goal node with the `«RuntimeAssumptionMonitor»` stereotype and a description placeholder (e.g., `monitor_expr="..."`). Each monitor is also assigned a `monitor_id` to link GSN nodes with runtime logs and mitigation actions. When a violation occurs, the corresponding node is marked with `«Reopened»` and populated with a runtime log or mitigation response.

In parallel, *confidence monitors* track a numerical "confidence score" for each GSN goal, inspired by dynamic assurance case approaches that embed probabilistic reasoning in structured arguments. Prior work maps safety/assurance arguments to Bayesian belief networks for evidence updating [10], applying Dempster–Shafer belief functions to combine heterogeneous evidence [22], and using Baconian-style eliminative reasoning to quantify confidence [24]. Here, confidence measures are used as heuristic indicators to prioritise review and regeneration activities rather than as definitive measures of system safety.

In our model, each confidence monitor is represented by annotating a GSN goal node with the `«ConfidenceMonitor»` stereotype and a description placeholder (e.g., `confidence_threshold="0.95"`) (Table 1). If confidence drops below this threshold, the goal is reopened and marked with `«DeferredEvidence»` for review. Monitor outputs (e.g., events, logs, confidence scores) are trace-linked to corresponding GSN nodes, maintaining consistency between runtime evidence

and the assurance argument. This structured use of placeholders and stereotypes ensures the assurance case remains continuously updated—reinforcing, weakening, or reopening claims in response to operational conditions.

Evolution-time assurance (change management and regeneration) Evolution-time assurance focuses on updating the assurance case when system changes or runtime violations occur. At evolution-time, changes to the system software, configuration, or operating assumptions trigger structured regeneration of affected assurance case arguments. While activities such as impact analysis and evidence planning are conceptually related to design-time reasoning, in this framework they are triggered by post-deployment changes and are therefore placed within the evolution-time phase. The evolution-time phase adopts a structured change management process inspired by ReASSURE [17]. When system changes arise—such as software updates, reconfigurations, feature additions, or persistent runtime alerts—GSN nodes marked with `«DeferredEvidence»` or `«ConfidenceMonitor»` serve as entry points for analysis (Table 1). This is supported by assembling an *evolution package* comprising model deltas, monitor logs, incident reports, and reopened GSN goals. Each opened goal carries an `evidence_cost="..."` tag from design-time to estimate regeneration effort. Using the `evidence_cost` tags and trace links from each GSN node to its originating RoboChart or PRISM artefact, we generate an `«ImpactAnalysis»` summary classifying goals as *valid*, *invalid*, or *uncertain*. Invalid or uncertain goals become candidates for regeneration. During the evidence planning phase, we prioritise these based on `evidence_cost` and safety criticality, and annotate each with a `«RegenerationPlan»` stereotype describing whether proof, simulation, or manual review is required.

Evidence regeneration can be performed using automated tools (e.g., rerunning CSP/FDR4 checks or PRISM analyses) for low-cost goals, while complex or high-impact goals require manual modelling and review. Upon successful regeneration, each affected GSN node's placeholder is updated with new evidence, its stereotype changes from `«DeferredEvidence»` to `«EvidenceProvided»`, and its version tag is incremented. Runtime monitor thresholds (from `«RuntimeAssumptionMonitor»` stereotypes) are also restored or adjusted based on the new evidence.

This closes the feedback loop, ensuring that evolution-time activities resolve the placeholders and stereotypes introduced at design-time and maintain alignment between the assurance case and the evolving autonomous system. Trace links between models, verification results, and GSN nodes ensure consistent propagation of changes across the assurance lifecycle. Together, these phases form a lifecycle-aware process in which design-time evidence seeds the assurance case, runtime monitoring updates it, and evolution-time processes regenerate affected arguments. Our framework supports partially automated assurance case maintenance across the lifecycle by unifying specifications, verification results, and assurance case regeneration in a single workflow. This enables a consistent single source of truth from the formal model to the structured argument, while reserving manual intervention for high-stakes scenarios. Although demonstrated

Table 1. Use of placeholders and stereotypes across design-time (D), runtime (R), evolution-time (E) assurance, and all three phases (A).

Name	Phase	Description
Placeholders		
`trace_expr="..."`	D, R	Trace-based assumptions for monitor configuration and violation detection.
`monitor_id="..."`	A	Identifier linking GSN goal to runtime monitor and logs.
`deferred=true`	A	Indicates that a claim is not yet discharged and requires runtime or post-change validation.
`confidence_-threshold="..."`	R	Minimum accepted confidence score before a goal is reopened.
`monitor_expr="..."`	R	Specifies the runtime condition or logic checked by a monitor.
`evidence_-cost="..."`	D, E	Time/effort estimate to regenerate evidence, used for prioritisation.
`evolution_-package="..."`	E	Contains triggering artefacts (e.g., logs, diffs, reports).
`impact_-summary="..."`	E	Classifies analysis results (valid, invalid, uncertain).
`regeneration_-plan="..."`	E	Specifies the repair strategy (e.g., simulation, proof, manual review).
Stereotypes		
`«TraceMonitored»`	D, R	Identifies goals linked to runtime-monitored trace assumptions.
`«DeferredEvidence»`	A	Flags goals where evidence is postponed, missing, or reopened.
`«RuntimeAssumption Monitor»`	R, E	Identifies runtime assumption monitors linked to control or sensor assumptions.
`«ConfidenceMonitor»`	R, E	Tracks probabilistic or confidence-based monitors.
`«Reopened»`	R, E	Marks goals reopened due to threshold violation or runtime alerts.
`«RegenerationPlan»`	E	Indicates the plan to recover missing evidence.
`«ImpactAnalysis»`	E	Tags nodes summarising the outcome of change analysis.
`«EvidenceProvided»`	E	Confirms goal closure with new evidence after regeneration.

on a robotics case study, the framework is designed to be applicable to other autonomous systems.

4 Case Study: An Illustrative Scenario of a Nuclear Radiation Inspection Robot

We illustrate our continuous assurance approach through a case study involving a *Nuclear Radiation Inspection Robot*. An unmanned ground vehicle (UGV), such as a Scout Mini, patrols a sequence of four predefined waypoints (e.g., l_3, l_4, l_1, and l_2) within a nuclear storage facility. At each stop, the UGV measures ambient radiation levels, timestamps dose-rate readings, and transmits data to a remote monitoring station. The UGV is equipped with LiDAR for obstacle detection,

depth cameras for localisation, radiation sensors for dose rate measurement, and onboard diagnostics for battery monitoring.

To ensure mission integrity and safety, the UGV must: (a) follow a strict, ordered patrol route without deviation (e.g., $l_3 \rightarrow l_4 \rightarrow l_1 \rightarrow l_2$); (b) avoid restricted or "no-go" zones (e.g., known radiation hot spots); and (c) terminate the mission early if battery charge falls below a safe operational threshold. To enforce these requirements under elevated radiation, a three-state *safety-wrapper controller* supervises the UGV's operational mode—`Patrol`, `Caution`, and `Emergency Retrieval`. Ambient dose-rate readings are evaluated against thresholds R_1 and R_2. Exceeding R_1 switches the controller to `Caution` mode and slows the UGV; exceeding R_2 triggers an immediate transition to `Emergency Retrieval`. In `Caution` mode, the robot proceeds at reduced speed to limit radiation exposure—by reducing time spent in high-radiation regions and lowering the likelihood of entering hazardous zones—while also conserving energy. In `Emergency Retrieval` mode, the robot halts in place (absorbing behaviour) and awaits recovery or manual retrieval. Battery depletion below the defined threshold in any mode triggers an `Abort` state, ending the mission safely. In both `Caution` and `Emergency Retrieval` modes, operator or navigator inputs are ignored to prevent unsafe overrides. This scenario serves as the common reference model for both our RoboChart and PRISM verification workflows, supporting formal analysis of functional correctness, safety guarantees, and probabilistic mission outcomes.

5 Formal Verification using RoboChart and PRISM

In this section, we describe the design-time phase of our framework. We present two *formal verification* workflows—RoboChart/FDR4 for functional analysis and PRISM for probabilistic analysis—and show how assurance case arguments are regenerated when requirements or verification results change. In this work, safety is characterised through qualitative functional and safety guarantees established via RoboChart/FDR4 verification, together with quantitative probabilistic and reward-based results derived from PRISM analyses.

Formal verification is a central element of our assurance methodology, integrated into the development process under a *design-for-assurance* paradigm, providing mathematical evidence that critical safety properties are satisfied by design before deployment. Unlike simulation or testing—which explore only a finite subset of behaviours—formal verification techniques such as model checking examine all reachable executions defined by a model's formal semantics. This is particularly important for autonomous systems, where uncommon environmental conditions and rare concurrency interleavings can trigger failures missed by conventional testing. Our verification approach employs a *heterogeneous verification* strategy [15] using two complementary toolchains: RoboChart with FDR4 for refinement-based functional correctness, and PRISM for analysis under probabilistic uncertainty. Together, these enable both verification of safety

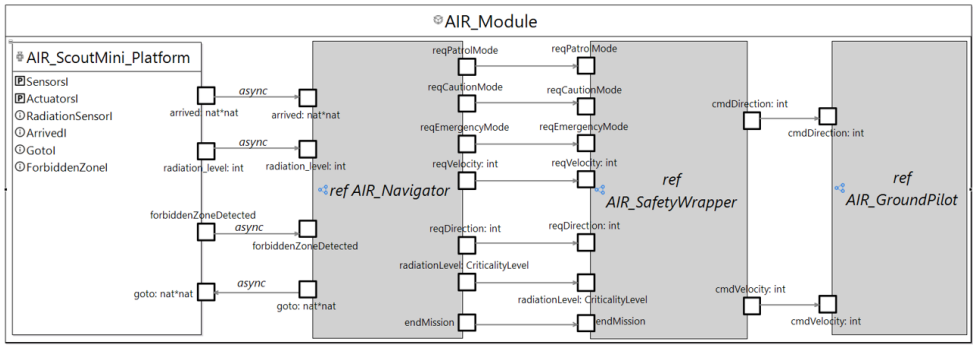

Fig. 2. High-level architecture of the nuclear inspection robot in the case study.

properties over all executions and quantitative assessment of undesirable events under realistic operational assumptions.

5.1 RoboChart Modelling of the Case Study

We model the core safety behaviour of the inspection robot's control architecture using RoboChart [14, 16], a domain-specific modelling language based on state machines with CSP-defined semantics. RoboChart models are organised into modules containing a robotic platform and one or more concurrent controllers. Using RoboTool, RoboChart specifications are translated into CSP under tock-CSP semantics [16], enabling timed refinement checking in FDR4.

In the case study (Section 4), we model the high-level safety architecture of an unmanned ground vehicle (UGV) for nuclear radiation inspection. Figure 2 shows the architecture comprising three controllers: `AIR_Navigator` (issuing high-level motion commands and detecting radiation levels), `AIR_SafetyWrapper` (responsible for enforcing radiation-aware safety policies), and `AIR_GroundPilot` (responsible for executing approved commands on the robot's actuators). `AIR_ScoutMini_Platform` acts as the robotic platform, providing a hardware abstraction layer for sensors and actuators. The `AIR_SafetyWrapper` acts as a mediating safety layer: it intercepts operator (or navigator) commands, evaluates them against the current hazard mode, and either forwards or suppresses them. Although AIR_SafetyWrapper is shown as a separate controller in Fig. 2, it logically mediates between navigation decisions and actuation. This centralisation of safety logic enforces a clear separation of concerns and facilitates traceability from requirements to model elements.

At the core of the `AIR_SafetyWrapper` is a hierarchical three-mode state machine (Fig. 3) that adapts to ambient radiation levels. In *Patrol Mode (PM)*, the robot follows a waypoint sequence at full speed while blocking entry into unsafe zones. When the dose exceeds threshold R_1, the wrapper transitions to *Caution Mode (CM)*, reducing speed and raising alerts. If the dose exceeds the critical threshold R_2, the system switches to *Emergency Retrieval Mode (ERM)*, an absorbing safety state where the robot halts in place and awaits recovery or

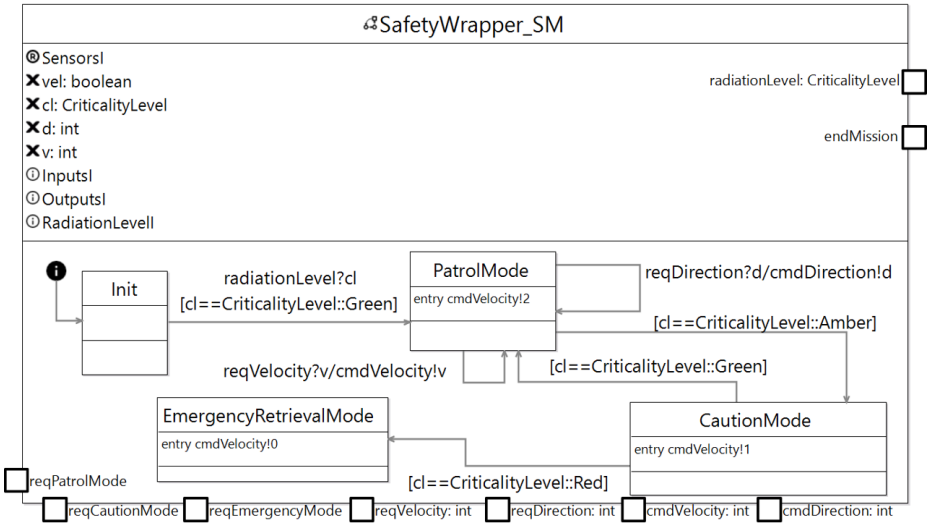

Fig. 3. State machine model of the safety wrapper controller in the case study scenario.

manual retrieval. The speed adjustments are executed by the `AIR_GroundPilot` controller via the robot's actuators.

The `AIR_Navigator` controller comprises two state machines: `Navigation_SM` (Fig. 4), which governs waypoint progression, and `RadiationLevelMonitor_SM`, which maps sensor readings to hazard levels for the safety wrapper. `Navigation_SM` defines waypoint-based mission behaviour, including progression through four sequential waypoint states (`W3`, `W4`, `W1`, `W2`) and safety constraints that may terminate the mission early. Transitions are guarded by arrival events and battery thresholds; at each transition, the robot issues a `goto` command to the next waypoint and decrements the battery level. In addition to this nominal route, `Navigation_SM` introduces safety-related terminal states: `Success` (mission completion after reaching the final waypoint), `ForbiddenFailure` (triggered by entry into restricted zones), and `Abort` (triggered by insufficient battery charge). `RadiationLevelMonitor_SM` models the robot's handling of radiation sensor inputs and their translation into abstract hazard levels for the safety wrapper (i.e., `Green` (safe), `Amber` (warning), and `Red` (critical)). The `AIR_Navigator`, `AIR_SafetyWrapper`, and `AIR_GroundPilot` processes are composed in parallel, synchronising on shared channels to form a closed-loop control model.

System requirements are formalised as CSP refinement assertions over observable controller events and incorporated into the CSP specifications generated by RoboTool. These assertions capture high-level safety and liveness requirements. For example, they ensure that the system maintains progress, follows its strict patrol route, avoids forbidden zones, responds appropriately to radiation thresholds, blocks unsafe inputs, and adheres to battery and emergency constraints. Any violation produces a counterexample trace that identifies the specific sequence of events leading to failure. Below, we present several key assertions for the `SafetyWrapper_SM` state machine and illustrate one in Listing 1.1.

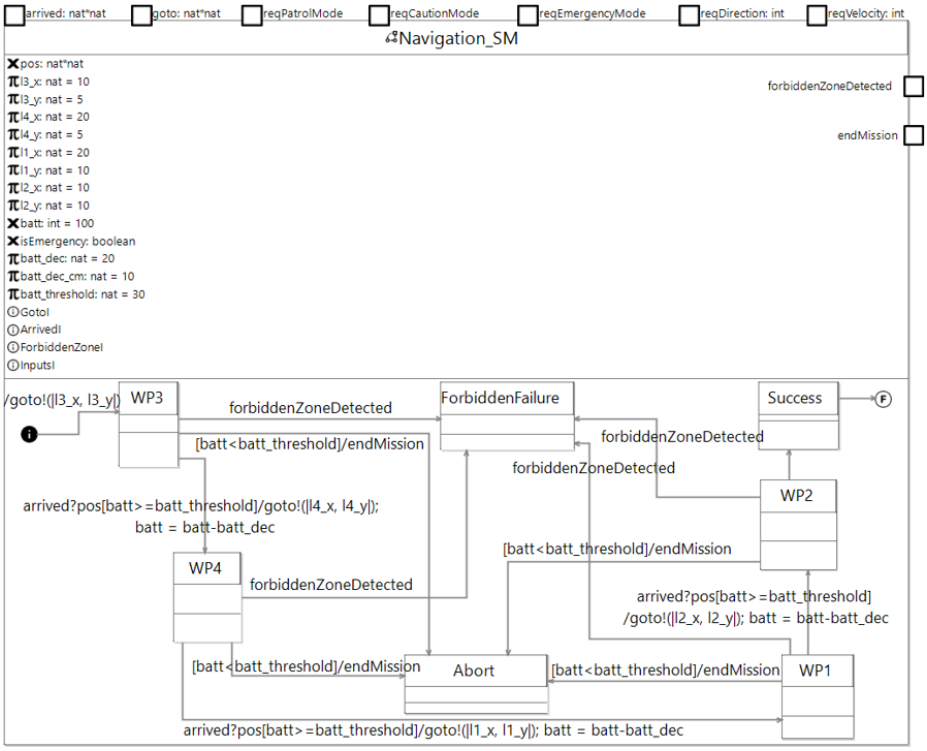

Fig. 4. Navigation state machine model for waypoint navigation in the case study.

- **A1-2_Deterministic-NoDeadlocks**: The system (e.g., `SafetyWrapper_SM`) must be deterministic and never enter a deadlocked state.
- **A3_WaypointSeq**: The UGV visits waypoints in a strict sequence.
- **A4_ModeSwitch**: Correct mode transitions in response to radiation readings.
- **A5_Velocity**: Immediate velocity adjustment upon hazard detection.
- **A6_NoReq**: Operator and navigator inputs are blocked during unsafe modes (CM or ERM).
- **A7_ERMAbsorb**: Once in Emergency Retrieval Mode, the system remains there until mission termination.
- **A8_BattAbort**: If battery charge falls below the configured threshold, the system transitions immediately to `Abort` and ends the mission.

Mode–switch correctness under radiation readings `A4_ModeSwitch`, a CSP refinement assertion, ensures that each radiation reading triggers the appropriate speed command by enforcing the mapping between detected radiation levels and commanded velocities (Listing 1.1).

```
csp ModeSwitchSpec associated to SafetyWrapper_SM
csp-begin ModeSwitchSpec =
SafetyWrapper_SM::radiationLevel.in?cl ->
```

```
( if cl == CriticalityLevel_Green
then SafetyWrapper_SM::cmdVelocity.out!2 -> ModeSwitchSpec
else if cl == CriticalityLevel_Amber
then SafetyWrapper_SM::cmdVelocity.out!1 -> ModeSwitchSpec
else
SafetyWrapper_SM::cmdVelocity.out!0 -> ModeSwitchSpec )
|~| SafetyWrapper_SM::reqDirection.in?d   -> ModeSwitchSpec
|~| SafetyWrapper_SM::reqPatrolMode.in    -> ModeSwitchSpec
|~| SafetyWrapper_SM::endMission.in       -> ModeSwitchSpec
|~| SafetyWrapper_SM::reqCautionMode.in   -> ModeSwitchSpec
|~| SafetyWrapper_SM::reqVelocity.in?v    -> ModeSwitchSpec
|~| SafetyWrapper_SM::reqEmergencyMode.in -> ModeSwitchSpec
|~| SafetyWrapper_SM::cmdDirection.out?d  -> ModeSwitchSpec
|~| SafetyWrapper_SM::terminate           -> SKIP
csp-end
untimed assertion A4_ModeSwitch: SafetyWrapper_SM refines
    ModeSwitchSpec in the traces model
```

Listing 1.1. Excerpt of the mode–switch specification and refinement assertion.

5.2 PRISM Modelling and Verification of the Case Study

The PRISM model complements the RoboChart analysis by capturing stochastic aspects of system behaviour that are not represented in deterministic models. While RoboChart and FDR4 provide exhaustive guarantees of logical and real-time correctness, they do not account for stochastic disturbances, including random collisions, radiation spikes, and energy depletion. To address this, we construct a PRISM [18] Discrete-Time Markov Chain (DTMC) model parameterised by environmental factors such as the probability of entering a forbidden zone (p_{err}) and radiation event rates. This enables both qualitative and quantitative verification under probabilistic behaviour. Our current PRISM model comprises two synchronised modules: `AIR_Navigator`, capturing mission logic including waypoint navigation, battery usage, and radiation sampling; and `AIR_SafetyWrapper` enforcing runtime safety policies through a three-mode supervisory controller. The wrapper regulates both the robot's velocity and operational mode based on sensed radiation levels. The PRISM model of the case study is provided in Appendix A.

The robot navigates four sequential waypoints (`loc` = 0–3), with terminal states `loc` = 4 (success), `loc` = 5 (forbidden-zone entry), and `loc` = 6 (battery abort or completed emergency retrieval). These PRISM locations correspond to the waypoint sequence $l_3 \to l_4 \to l_1 \to l_2$ used in the RoboChart model. Radiation levels are discretised as Safe (0), Warning (1), and Critical (2). Battery state-of-charge (`batt`) ranges from 0 to 100 and decreases with each movement. The control mode variable `sw` encodes the current safety state (0 = Patrol, 1 = Caution, 2 = Emergency Retrieval). The `AIR_Navigator` module governs waypoint transitions, where each step consumes energy, carries a forbidden-zone

Table 2. PCTL and reward-based properties for the PRISM model of the case study.

Property (PCTL/Reward)	**Description**
Navigation Properties	
$P =? [F \text{ loc} = 4]$	Probability mission eventually succeeds
$P =? [F \text{ loc} = 5]$	Probability robot eventually enters a forbidden zone
$P =? [G \text{ loc} \neq 5]$	Probability mission always avoids a forbidden zone
$P =? [(\text{loc} \neq 5)\ U\ (\text{loc} = 4)]$	Probability goal reached without entering a forbidden zone
$P =? [F \text{ batt} < \text{batt_threshold}]$	Probability battery eventually drops below safety threshold
$P =? [F <= 5 \text{ loc} = 4]$	Probability mission completes within five steps
$P =? [G (\text{loc} \neq 5 \ \&\ \text{loc} \neq 6 \ \&\ \text{batt} \geq \text{batt_threshold})]$	Probability robot always remains safe and above energy threshold
$R\{\text{"dose"}\} =? [F (\text{loc} = 4 \mid \text{loc} = 5 \mid \text{loc} = 6)]$	Expected cumulative radiation exposure events before mission termination
$R\{\text{"moves"}\} =? [F (\text{loc} = 4 \mid \text{loc} = 5 \mid \text{loc} = 6)]$	Expected total navigation moves until absorption (success, failure, or abort)
$R\{\text{"time_in_cm"}\} =? [F (\text{loc} = 4 \mid \text{loc} = 5 \mid \text{loc} = 6)]$	Expected cumulative time steps in Caution mode before mission termination
$R\{\text{"time_stopped"}\} =? [F (\text{loc} = 4 \mid \text{loc} = 5 \mid \text{loc} = 6)]$	Expected cumulative time steps halted in Emergency Retrieval mode
Safety Wrapper Properties	
$P \geq 1 [F (\text{rad} = 1 \ \&\ \text{sw} = 1)]$	Warning-level radiation always triggers entry into Caution mode
$P \geq 1 [F (\text{rad} = 2 \ \&\ \text{sw} = 2)]$	Critical-level radiation always triggers entry into Emergency Retrieval mode
$P \geq 1 [G (\text{sw} = 0 \rightarrow \text{vel} = 2)]$	Patrol mode always enforces full velocity
$P \geq 1 [G (\text{sw} = 1 \rightarrow \text{vel} = 1)]$	Caution mode always enforces reduced velocity
$P \geq 1 [G (\text{sw} = 2 \rightarrow \text{vel} = 0)]$	Emergency Retrieval mode always enforces a complete stop
$P \leq 0 [F (\text{sw} \neq 0 \ \&\ \text{op_used})]$	Operator input never occurs outside Patrol mode

entry risk (`p_err`), and samples radiation probabilistically. If `batt` falls below a defined threshold, the robot transitions to the abort state.

The `AIR_SafetyWrapper` manages mode transitions in response to environmental conditions. Warning-level radiation triggers a transition from `Patrol` to `Caution`, reducing speed. Critical radiation immediately switches the system to `Emergency Retrieval`, an absorbing state in which the robot halts and awaits recovery or manual retrieval. Once entered, the robot remains in safety override mode. Operator inputs (via synchronised labels `[hdng]` and `[vel]`) are permitted only in `Patrol` mode and tracked using the Boolean variable `op_used`. To assess the system quantitatively, we define several reward structures: `moves` (forward progress), `dose` (high-radiation exposures), `time_in_cm` (duration in `Caution` mode), and `time_stopped` (halted states in `Emergency Retrieval`). These allow cumulative tracking of operational behaviours relevant to the mission and safety assurance. Probabilistic and reward-based PCTL properties (see Table 2 and Fig. 5) are used to verify correctness, assess risk, and confirm enforcement of key safety policies at runtime.

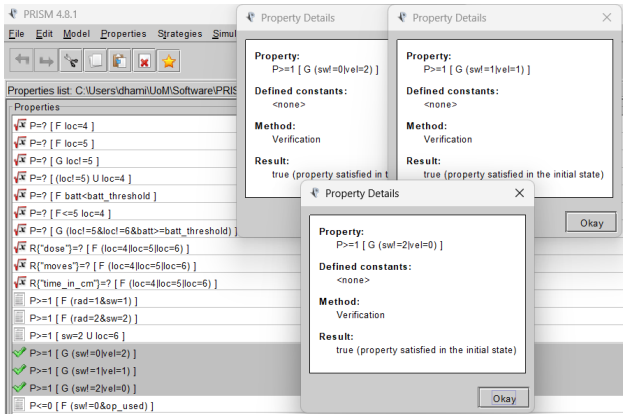

Fig. 5. Formal verification of PRISM properties in the case study.

5.3 Transforming Formal Specification and Verification Results into Assurance Case Arguments

As autonomous systems evolve, structured assurance case arguments must remain synchronised with underlying formal specifications and associated verification results (design-time evidence). This traceability is essential, as manual regeneration of assurance case arguments risks inconsistency and transcription errors and can hinder agile development. Towards addressing this, we developed `PRISM2GSN`, an Eclipse plugin based on the Eclipse Rich Client Platform (RCP) and OSGi component model, which integrates the PRISM model checker into Eclipse and automates regeneration of GSN arguments from updated property specifications. The current implementation (available from [1]) supports one-way transformation from PRISM to GSN; bidirectional synchronisation remains future work.

`PRISM2GSN` monitors temporal logic property specifications in PRISM `.props` files, including probabilistic reachability, time-bounded, and reward-based properties. Whenever a property file is saved, a verification-to-assurance pipeline is triggered automatically: the relevant PRISM model is executed, results are parsed, and corresponding GSN assurance arguments are regenerated. The plugin supports a range of PCTL property types (time-bounded, conditional, and reward-based) and is extensible to accommodate new property classes.

The extracted results are transformed into a structured assurance argument in the domain-specific language (DSL) of the AdvoCATE assurance case tool [3,9]. Currently, a generated argument contains (i) a top-level goal asserting satisfaction of the verified property set; a strategy node describing decomposition by individual property; and one sub-goal per property, each linked to its formal specification (context) and PRISM verification result (solution). The generated DSL file can be directly visualised and edited in the AdvoCATE tool. After generating assurance arguments, structured placeholders and stereotypes are manually inserted into the GSN elements to support traceability and enable automated updates during runtime and post-deployment. Figure 6 shows part of

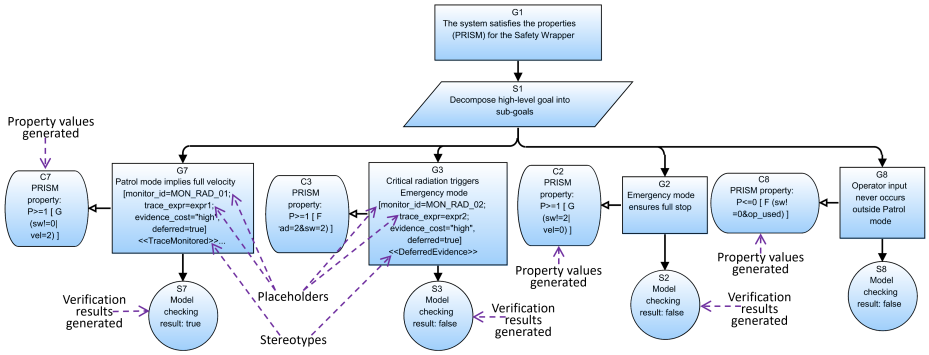

Fig. 6. Part of an assurance case argument generated by the PRISM2GSN plugin for the safety wrapper controller, with several placeholders and stereotypes manually inserted at design-time after generation.

an assurance case generated by the PRISM2GSN plugin for the safety wrapper, with several placeholders and stereotypes (e.g., `trace_expr`, `evidence_cost`, `«DeferredEvidence»`) manually inserted at design-time after generation.

6 Discussion: Regulatory Considerations

This section provides a discussion of our proposed approach, focusing on its alignment with regulatory expectations. The *Trilateral Principles* study [7] outlines regulatory expectations for the safe deployment of AI in nuclear applications across five key domains: safety and security engineering, human factors, AI architecture, lifecycle management, and assurance documentation.

Risk-Proportionate Design and Mode Stratification: The risk model in [7, Sec. 3] advocates proportionate assurance aligned with both autonomy level and safety consequence. Our three-mode `AIR_SafetyWrapper`—`Patrol`, `Caution`, and `Emergency Retrieval`—corresponds to Regions 2–4, representing increasing autonomy and consequence. `Patrol` mode supports routine low-dose surveying with full autonomy; `Caution` mode enforces speed limitations under moderate radiation; and `Emergency Retrieval` withdraws under high-dose detection.

Modular and Secure AI Architecture: Consistent with [7, Secs. 4,6], we adopt a modular design with clearly defined component boundaries and constrained input/output interfaces. All radiation-response logic is encapsulated within a standalone `AIR_SafetyWrapper` module that intercepts and regulates commands issued by the operator or the navigator. The module is formally verified for deadlock freedom, compliance with mode-transition deadlines, and enforcement of operational constraints. The architecture preserves separation of concerns by decoupling AI-based decision logic from actuation and communication subsystems, consistent with the report's call for design separation to enhance explainability, fault containment, and assurance traceability.

Lifecycle Oversight and Continuous Assurance: Lifecycle-spanning assurance [7, Sec. 7] motivates our model-driven pipeline (Section 3), which maintains

traceability between PRISM analyses and GSN assurance cases. Placeholders, monitor bindings, and evidence cost tags are included to support runtime updates and evolution-time regeneration. Derived runtime monitors will observe environmental assumptions (e.g., radiation bounds), and assurance monitors will track probabilistic confidence over time. Violations trigger GSN goal reopening and selective re-verification, consistent with the emphasis on configuration management, drift detection, and adaptive retraining [7, Sec. 7].

Human–AI Interaction and Organisational Factors: [7, Sec. 5] stresses the critical role of human–AI interaction, advocating for explainable interfaces, intervention pathways, and traceable control transitions. Handover events and override rules are explicitly modelled in RoboChart and verified to prevent unsafe operator actions during hazardous modes. The GSN model incorporates assumptions on manual intervention, and our verification ensures that no operator command can bypass `AIR_SafetyWrapper's` constraints in *Caution* or *Emergency Retrieval* modes.

Documented, Dynamic Safety Cases: Transparent and evolving assurance documentation [7, Sec. 8] aligns with our integration of formal verification artefacts into GSN structures so assurance claims remain traceable and updatable. Design-time claims include placeholders for runtime validation and evolution-time updates, annotated with effort cost and monitor identifiers; violated assumptions or reduced confidence automatically reopen affected goals, with regeneration prioritised by cost and safety criticality.

7 Conclusion

In this paper, we proposed a unified *Continuous Assurance Framework* that integrates design-time, runtime, and evolution-time assurance activities into a model-driven, lifecycle-aware methodology. Through an illustrative case study involving an autonomous nuclear inspection robot, we instantiated the design-time stage of the framework using dual formal verification workflows: RoboChart with FDR4 for functional correctness, and PRISM for probabilistic risk quantification. These complementary analyses were integrated into structured GSN arguments enriched with placeholders and stereotypes to support traceability and automated updates. We discussed the framework in relation to the *Trilateral AI Principles*, thus aligning with regulator-endorsed best practices.

The current implementation is limited to design-time automation and does not yet provide runtime monitoring or automated evolution-time regeneration. Extending the framework to incorporate runtime monitoring, evolution-time regeneration, and support for multi-agent settings is an important direction for future work. This includes implementing monitors to detect violations of design-time assumptions about sensor data or control loop behaviour, and updating assurance confidence scores based on real-time system data. We will investigate enhancing evolution-time tooling to support automated change impact analysis and selective regeneration of verification artefacts. We also aim to broaden the framework's applicability by incorporating learning-enabled components and

adaptive behaviours, thereby addressing emerging challenges in *assured autonomy* under epistemic uncertainty. Another potential direction involves integration with an assurance case management platform to improve tool interoperability across the verification-to-certification pipeline.

Acknowledgments. This work is supported by the *Centre for Robotic Autonomy in Demanding and Long Lasting Environments* (EPSRC grant EP/X02489X/1) and by the Royal Academy of Engineering under the *Chairs in Emerging Technology* scheme.

References

1. Abeywickrama, D.B.: PRISM2GSN: Eclipse Plugin for Transforming Specifications and Verification Results into GSN Argument Models. https://github.com/DhamindaA/prism2gsn-eclipse-plugin (Aug 2025)
2. Abeywickrama, D.B., Bennaceur, A., Chance, G., Demiris, Y., Kordoni, A., Levine, M., Moffat, L., Moreau, L., Mousavi, M.R., Nuseibeh, B., Ramamoorthy, S., Ringert, J.O., Wilson, J., Windsor, S., Eder, K.: On specifying for trustworthiness. Commun. ACM **67**(1), 98–109 (Dec 2023). https://doi.org/10.1145/3624699, https://doi.org/10.1145/3624699
3. AdvoCATE: User Guide AdvoCATE (Version 1.10) (Jan 2023)
4. Belle, A.B., Hemmati, H., Lethbridge, T.C.: Position paper: A vision for the dynamic safety assurance of ML-enabled autonomous driving systems. In: 2023 IEEE 31st International Requirements Engineering Conference Workshops (REW). pp. 297–301 (2023). https://doi.org/10.1109/REW57809.2023.00056
5. Bourbouh, H., Farrell, M., Mavridou, A., Sljivo, I., Brat, G., Dennis, L.A., Fisher, M.: Integrating formal verification and assurance: An inspection rover case study. In: NASA Formal Methods: 13th International Symposium, NFM 2021, Virtual Event, May 24–28, 2021, Proceedings. p. 53–71. Springer-Verlag, Berlin, Heidelberg (2021). https://doi.org/10.1007/978-3-030-76384-8_4, https://doi.org/10.1007/978-3-030-76384-8_4
6. Calinescu, R., Češka, M., Gerasimou, S., Kwiatkowska, M., Paoletti, N.: Efficient synthesis of robust models for stochastic systems. Journal of Systems and Software **143**, 140–158 (2018). https://doi.org/10.1016/j.jss.2018.05.013
7. Canadian Nuclear Safety Commission, UK Office for Nuclear Regulation, US Nuclear Regulatory Commission: Considerations for developing artificial intelligence systems in nuclear applications. Trilateral principles report, Canadian Nuclear Safety Commission; UK Office for Nuclear Regulation; US Nuclear Regulatory Commission (September 2024)
8. Cârlan, C., Gallina, B., Soima, L.: Safety case maintenance: A systematic literature review. In: Habli, I., Sujan, M., Bitsch, F. (eds.) Computer Safety, Reliability, and Security. pp. 115–129. Springer International Publishing, Cham (2021)
9. Denney, E., Lee, R., Pai, G.J., Šljivo, I.: QUASAR: Quantifiable assurance cases for trusted autonomy. Tech. Rep. AFRL-RI-RS-TR-2023-162, Air Force Research Laboratory Information Directorate (2023)
10. Denney, E., Pai, G., Habli, I.: Towards measurement of confidence in safety cases. In: Proceedings of the 2011 International Symposium on Empirical Software Engineering and Measurement. p. 380–383. ESEM '11, IEEE Computer Society, USA (2011). https://doi.org/10.1109/ESEM.2011.53, https://doi.org/10.1109/ESEM.2011.53

11. Dong, Y., Huang, W., Bharti, V., Cox, V., Banks, A., Wang, S., Zhao, X., Schewe, S., Huang, X.: Reliability assessment and safety arguments for machine learning components in system assurance. ACM Trans. Embed. Comput. Syst. **22**(3) (Apr 2023). https://doi.org/10.1145/3570918, https://doi.org/10.1145/3570918
12. Hartsell, C., Ramakrishna, S., Dubey, A., Stojcsics, D., Mahadevan, N., Karsai, G.: Resonate: A runtime risk assessment framework for autonomous systems. In: 2021 International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 118–129 (2021). https://doi.org/10.1109/SEAMS51251.2021.00025
13. Hoare, C.A.R.: An axiomatic basis for computer programming. Commun. ACM **12**(10), 576–580 (Oct 1969). https://doi.org/10.1145/363235.363259, https://doi.org/10.1145/363235.363259
14. Li, W., Ribeiro, P., Miyazawa, A., Redpath, R., Cavalcanti, A., Alden, K., Woodcock, J., Timmis, J.: Formal design, verification and implementation of robotic controller software via robochart and robotool. Auton. Robots **48**(6) (Jul 2024). https://doi.org/10.1007/s10514-024-10163-7, https://doi.org/10.1007/s10514-024-10163-7
15. Luckcuck, M., Farrell, M., Dennis, L.A., Dixon, C., Fisher, M.: Formal specification and verification of autonomous robotic systems: A survey. ACM Comput. Surv. **52**(5) (Sep 2019). https://doi.org/10.1145/3342355, https://doi.org/10.1145/3342355
16. Miyazawa, A., Ribeiro, P., Li, W., Cavalcanti, A., Timmis, J., Woodcock, J.: RoboChart: modelling and verification of the functional behaviour of robotic applications. Software and Systems Modeling **18**(5), 3097,3149 (Jul 2019). https://doi.org/10.1007/s10270-018-00710-z
17. Murphy, L., Carwehl, M., Päßler, J., Chechik, M.: Towards systematic maintenance of assurance for evolving self-adaptive systems. In: Proc. of the Engineering Reliable Autonomous Systems Conference (ERAS 2025). IEEE (2025)
18. PRISM Model Checker: Property Manual. Online (2026), https://www.prismmodelchecker.org/manual/Main/AllOnOnePage
19. Safety-Critical Systems Club: Goal Structuring Notation. https://scsc.uk/gsn (2024)
20. Rouff, C., Watkins, L.: Assured autonomy survey. Foundations and Trends in Privacy and Security **4**(1), 1–116 (2022). https://doi.org/10.1561/3300000027, http://dx.doi.org/10.1561/3300000027
21. Schleiss, P., Carella, F., Kurzidem, I.: Towards continuous safety assurance for autonomous systems. In: 2022 6th International Conference on System Reliability and Safety (ICSRS). pp. 457–462 (2022). https://doi.org/10.1109/ICSRS56243.2022.10067323
22. Wang, R., Guiochet, J., Motet, G., Schön, W.: Safety case confidence propagation based on dempster–shafer theory. International Journal of Approximate Reasoning **107**, 46–64 (2019). https://doi.org/https://doi.org/10.1016/j.ijar.2019.02.002, https://www.sciencedirect.com/science/article/pii/S0888613X18303505
23. Wei, R., Jiang, Z., Mei, H., Barmpis, K., Foster, S., Kelly, T., Zhuang, Y.: Automated model-based assurance case management using constrained natural language. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **43**(1), 291–304 (2024). https://doi.org/10.1109/TCAD.2023.3303220
24. Weinstock, C.B., Goodenough, J.B., Klein, A.Z.: Measuring assurance case confidence using baconian probabilities. In: 2013 1st International Workshop on Assurance Cases for Software-Intensive Systems (ASSURE). pp. 7–11 (2013). https://doi.org/10.1109/ASSURE.2013.6614264

A PRISM Formal Model for the Case Study

```
// ------------------------------------------------------------
// PRISM DTMC Model for Nuclear Radiation Inspection Robot with Safety-Wrapper
// ------------------------------------------------------------

dtmc
// ------------------------------------------------------------
// These constants define the environment and robot behavior parameters.

// Radiation sampling probabilities (must sum to 1.0)
const double p_rad_crit   = 0.02; // Probability of critical radiation (rad = 2)
const double p_rad_med    = 0.08; // Probability of warning radiation (rad = 1)
const double p_rad_safe   = 1 - p_rad_crit - p_rad_med; // Probability of safe radiation (rad = 0)

// Collision and battery dynamics
const double p_err        = 0.01; // Probability of entering a forbidden zone on any move
const int    batt_dec     = 20;  // Battery drain per move in Patrol mode
const int    batt_dec_cm  = 10;  // Battery drain per move in Caution mode (slower speed)
const int    batt_threshold = 30; // Minimum battery level (SoC %) to safely continue mission

// Radiation thresholds
const int    R1 = 1; // Warning radiation threshold (triggers Caution mode)
const int    R2 = 2; // Critical radiation threshold (triggers Emergency Retrieval mode)

// ------------------------------------------------------------
// Logical formulas used for guards and transitions
// ------------------------------------------------------------
formula is_warning  = (rad = 1); // true if Warning level radiation is sampled
formula is_critical = (rad = 2); // true if Critical level radiation is sampled

// ------------------------------------------------------------
// Module: AIR_Navigator
// Models the UGV's patrolling behavior
// ------------------------------------------------------------
module AIR_Navigator

// State variables
loc  : [0..6] init 0;     // Current location: 0-3 (waypoints), 4=Success, 5=Forbidden zone entry, 6=Abort
batt : [0..100] init 100; // Battery level (state of charge), initialized to 100%
rad  : [0..2] init 0;     // Most recent radiation sample: 0=Safe, 1=Warning, 2=Critical

// ------------------------------------------------------------
// Battery-abort transition: if battery is below threshold mid-mission
// ------------------------------------------------------------
[] loc < 4 & batt < batt_threshold ->
(loc' = 6) & (batt' = batt) & (rad' = rad);

// ------------------------------------------------------------
// Movement in Patrol Mode (sw = 0)
// Full speed, full battery drain, samples radiation at new location
// ------------------------------------------------------------
[] loc < 4 & batt >= batt_threshold & batt >= batt_dec & sw = 0 ->
p_err :
(loc' = 5) & (batt' = batt - batt_dec) & (rad' = rad) +
(1 - p_err) * p_rad_crit :
(loc' = loc + 1) & (batt' = batt - batt_dec) & (rad' = 2) +
(1 - p_err) * p_rad_med :
(loc' = loc + 1) & (batt' = batt - batt_dec) & (rad' = 1) +
(1 - p_err) * p_rad_safe :
(loc' = loc + 1) & (batt' = batt - batt_dec) & (rad' = 0);

// ------------------------------------------------------------
// Movement in Caution Mode (sw = 1)
```

```
// Slower motion, reduced battery drain, continues route cautiously
// ---------------------------------------------------------
[] loc < 4 & batt >= batt_threshold & batt >= batt_dec_cm & sw = 1 ->
p_err :
(loc' = 5) & (batt' = batt - batt_dec_cm) & (rad' = rad) +
(1 - p_err) * p_rad_crit :
(loc' = loc + 1) & (batt' = batt - batt_dec_cm) & (rad' = 2) +
(1 - p_err) * p_rad_med :
(loc' = loc + 1) & (batt' = batt - batt_dec_cm) & (rad' = 1) +
(1 - p_err) * p_rad_safe :
(loc' = loc + 1) & (batt' = batt - batt_dec_cm) & (rad' = 0);

// ---------------------------------------------------------
// Emergency Retrieval Mode (sw = 2)
// Halt in place, resets radiation to safe and no battery consumption
// ---------------------------------------------------------
[] loc < 4 & sw = 2 & batt >= batt_threshold -> (loc' = loc) & (batt' = batt) & (rad' =
   rad);

// ---------------------------------------------------------
// Absorbing Terminal States
// No transitions out of these; mission ends
// ---------------------------------------------------------
[] loc = 4 -> (loc' = 4); // Mission success
[] loc = 5 -> (loc' = 5); // Forbidden zone entry
[] loc = 6 -> (loc' = 6); // Abort (battery or retreat)
endmodule

// ------------------------------------------------------------
// Module: AIR_SafetyWrapper
// Supervisory controller for mode switching, velocity setting, and operator input control
// ------------------------------------------------------------
module AIR_SafetyWrapper

sw       : [0..2] init 0;    // Mode: 0=Patrol, 1=Caution, 2=Emergency Retrieval
vel      : [0..2] init 2;    // Velocity: 2=Full, 1=Slow, 0=Stopped
op_used  : bool init false; // True if any operator command (hdng or vel) has been used

// Transition: Patrol? Caution on warning-level radiation
[] sw = 0 & is_warning ->
(sw' = 1) & (vel' = 1);

// Transition: Patrol/Caution? Emergency Retrieval on critical-level radiation
[] sw <= 1 & is_critical ->
(sw' = 2) & (vel' = 0);

// Stay in Emergency Retrieval mode (absorbing)
[] sw = 2 -> (sw' = 2) & (vel' = 0);

// ---------------------------------------------------------
// Operator/Navigator Commands
// These commands are only enabled when sw = 0 (Patrol mode)
// When triggered, they set op_used = true for later tracking
// ---------------------------------------------------------
[hdng] sw = 0 -> (sw' = sw) & (op_used' = true); // heading request
[vel]  sw = 0 -> (sw' = sw) & (op_used' = true); // velocity command
endmodule

// ------------------------------------------------------------
// Reward Structures
// Used for quantitative analysis: cost, time, safety, and diagnostics
// ------------------------------------------------------------
// Reward: Each movement step forward (mission progress)
rewards "moves"
loc < 4 : 1;
endrewards
// Reward: Radiation exposure incidents (warning or critical)
rewards "dose"
```

```
rad >= 1 : 1;
endrewards
// Reward: Time spent in Caution mode (sw = 1)
rewards "time_in_cm"
sw = 1 : 1;
endrewards
// Reward: Time spent completely stopped (used for energy-saving diagnostics)
rewards "time_stopped"
vel = 0 : 1;
endrewards
```

Listing 1.2. PRISM DTMC model created for case study scenario.