VeriROS: Verifiable ROS2 Navigation Execution Framework

Huan Zhang
Maynooth University
Department of Computer Science
Maynooth, Ireland
huan.zhang.2024@mumail.ie

Hao Wu
Maynooth University
Department of Computer Science
Maynooth, Ireland
haowu@cs.nuim.ie

Abstract

We present VeriROS, a compact pipeline that issues machine-checkable *admission certificates* for ROS2 waypoint missions and ties those certificates to lightweight runtime evidence. At design time, a *Formal Gate* encodes conservative per-leg Worst-Case Execution Time (WCET) estimates and Processor Demand Criterion (PDC) feasibility windows into quantifier-free linear real arithmetic (QF_LRA) constraints; an SMT solver (Z3) then decides feasibility and yields either a certificate with minimum slack margins or an UNSAT core with repair hints. At runtime, missions execute under Earliest Deadline First (EDF) with safety fences while a ROS2 monitor evaluates STL robustness and empirical counters to validate, degrade, or revoke certificates. We describe an anonymised prototype and artefact bundle, outlining an evaluation plan. We conclude with open questions regarding certificate semantics, compositional PTA contracts, and scalability.

CCS Concepts

• **Software and its engineering → Software verification**.

Keywords

ROS2, admission certificate, probabilistic timed automata, runtime verification, STL, EDF, PDC, SMT

ACM Reference Format:
Huan Zhang and Hao Wu. 2026. VeriROS: Verifiable ROS2 Navigation Execution Framework. In *FormaliSE '26: International Conference on Formal Methods in Software Engineering, April 12–13, 2026, Rio de Janeiro, Brazil*. ACM, New York, NY, USA, 5 pages. https://doi.org/10.1145/3793656.3793685

1 Introduction

Autonomous robots built with ROS2 are increasingly deployed in time- and safety-critical domains (warehouses, logistics, last-mile delivery). These systems combine hard timing constraints (deadlines, speed limits, and safe stopping) with pervasive uncertainty from perception, localisation, and networking. Current engineering practice relies heavily on simulation and ad-hoc field testing; such empirical approaches are costly, provide limited machine-checkable evidence, and often fail to reveal subtle timing–probabilistic interactions until after deployment.

FormaliSE'26, Rio de Janeiro, Brazil

ACM ISBN 978-x-xxxx-xxxx-x/YYYY/MM
https://doi.org/10.1145/3793656.3793685

We propose *admission certificates*: compact, machine-checkable artefacts issued at design time that (i) record assumptions and verified slack/probabilistic margins, and (ii) remain meaningfully aligned with runtime evidence through a shared Probabilistic Timed Automata (PTA) semantic layer. Certificates act as auditable contracts that enable automated accept/degrade/revoke decisions during execution and support actionable repair suggestions when infeasibility is detected. We illustrate the idea with a lightweight pipeline (Extractor, Admission Gate, Runtime Monitor) and an anonymised ROS2 prototype and artefact bundle for reproducible demos.

Contributions. This research-ideas paper makes three contributions: (1) we introduce the *admission-certificate* abstraction and a minimal schema for binding PTA, verified properties, slack margins, and probabilistic bounds; (2) we describe a compact, reproducible prototype recipe (Extractor, Admission Gate, Runtime Monitor) and an anonymized artifact for evaluation; and (3) we identify three integration challenges (WCET vs. PDC complexity; compositional PTA contracts; robustness-to-slack mapping) and pose concrete research questions to stimulate community discussion.

2 Background and Motivation

We model waypoint missions as finite job sets $J = \{(r_i, C_i, d_i)\}$ where r_i is a release time, C_i a conservative Worst-Case Execution Time (WCET), and d_i a deadline. For single-processor preemptive scheduling, Earliest Deadline First (EDF) with the Processor Demand Criterion (PDC) yields exact feasibility checks for finite job sets [1, 8, 16]. Timed-automata (UPPAAL) and probabilistic model checkers (PRISM) enable expressive offline analyses [2, 5, 6]. Complementing offline tools, Signal Temporal Logic (STL) monitoring supplies online robustness semantics and sliding-window monitoring algorithms [3, 9, 11].

These tools are rarely combined in a single workflow. UPPAAL or PRISM may verify an abstract model, but the proof does not travel with the deployed binary; a field engineer cannot check whether the running code still matches the verified artefact. STL monitors detect violations at runtime yet give no advance warning that a mission is infeasible.

Admission certificates address this gap: they are small, structured artefacts that record what was verified, under what assumptions, and provide a runtime-checkable validity predicate that can trigger degradation or revocation.

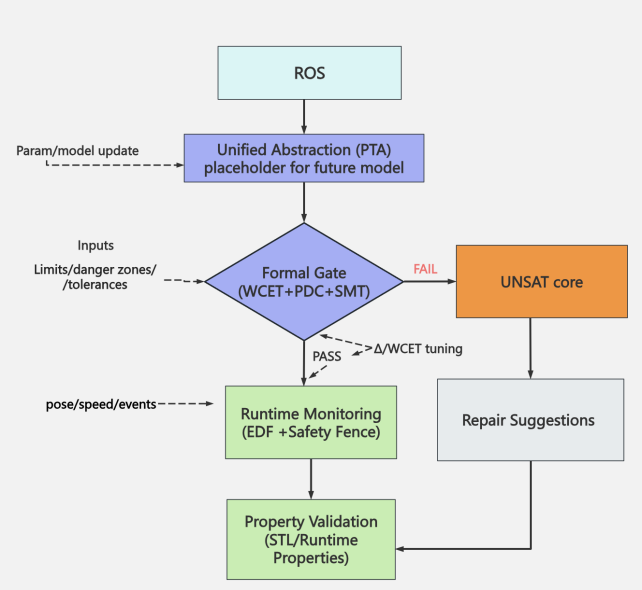

Figure 1: High-level VeriROS pipeline. Design-time: ROS → PTA → Formal Gate (WCET+PDC+SMT) issues admission certificates bound to PTA. Runtime: EDF execution with safety fences and online STL monitoring produces evidence tuples that feed the policy and repair pipeline.

3 Core Idea: Admission Certificates

3.1 Why admission certificates differ from existing approaches

Existing approaches include: (i) offline verification (UPPAAL/PRISM) that produces proofs or counterexamples but often lacks a tight binding to the exact runnable configuration; (ii) runtime monitoring that detects violations but generally offers no prior machine-checkable guarantees; (iii) narrative safety cases (DO-178C assurance artifacts) that are primarily written for human auditors and are not readily machine-checkable; and (iv) contract-based design which specifies interfaces but typically lacks concrete runtime evidence linkage.

Admission certificates are *compact, machine-checkable* artefacts and give us four advantages. (a) We can encode verified properties, such as safety invariants for scheduling and probabilistic bounds. (b) We can carry provenance and binding metadata (PTA, config checksum) for audit trails. (c) We could also revoke admission certificates when runtime checks violate safety invariants. (d) We can take advantage of SMT unsat cores to derive the hints for repairing certificates.

Figure 1 shows our VeriROS framework. The idea is to use PTA as a unified abstraction to encode scheduling properties of ROS. We then design a new component called Formal Gate that can issue an admission certificate when scheduling properties are successfully verified. If the verification is unsuccessful, we utilise returned unsat cores to extract repair hints to help users fix scheduling properties. The Formal Gate acts as a safety net for planning. Then the Runtime Monitoring and Property Validation components validate runtime events and produce traces that are machine checkable.

The PTA in Figure 1 is not invoked as a model-checking target in the current prototype. Instead, the Formal Gate encodes PDC constraints directly in QF_LRA and dispatches them to Z3. This choice is deliberate: standard PTA model checkers do not expose UNSAT cores, yet such cores are essential for the repair-hint mechanism demonstrated in Case B (Section 4). The PTA layer remains useful for certificate provenance (the `pta_hash` field binds a certificate to a specific abstraction) and provides a path toward richer stochastic analyses in future work.

3.2 Certificate Semantics: Syntax, Validity

We define the following compact certificate schema (in JSON format):

$$\text{Certificate} ::= \langle\ \text{PTA},\ \text{Properties},\ \text{Bindings},\ \text{Provenance}\ \rangle,$$

where PTA is a modular PTA encoding or a content hash, `Properties` lists verified items (schedulability with min_slack, safety invariants, probabilistic bounds), `Bindings` contains config checksums and event mappings, and `Provenance` records tool/version and `cert_quality`.

We define a conservative runtime validity predicate that covers both timing (STL) properties and probabilistic claims (counters):

$$\begin{aligned}\text{valid}(\text{cert}, \tau) \iff & \big(\text{Bindings.match}(\text{cert}, \text{runtime\_env}(\tau))\big) \\ & \wedge\ \big(\forall p \in \text{Properties}_{\text{timing}}.\ \rho_p(\tau) \geq \alpha \cdot \text{min\_slack}_p\big) \\ & \wedge\ \big(\forall q \in \text{Properties}_{\text{prob}}.\ \text{WilsonUpper}(s_q, n_q, 1-\gamma) \\ & \leq \text{bound}_q\big).\end{aligned}$$

A certificate c is *valid* at trace τ when three conditions hold: the configuration hash matches the running system, every timing property p has STL robustness $\rho_p(\tau) \geq \alpha \cdot \text{min\_slack}_p$, and every probabilistic claim q satisfies $\text{WilsonUpper}(s_q, n_q, 1-\gamma) \leq \text{bound}_q$. The parameter $\alpha \in (0, 1]$maps continuous robustness to scheduling slack; γ=0.05 gives 95% confidence for Wilson intervals [15].

If any condition fails for k consecutive observations or persists for duration Δ_{rev}, the certificate is revoked and the policy layer triggers a safe-stop or re-certification request.

We allow certificate revoke when $\exists\ p$ the property p violates the above conditions (violation for k consecutive samples or over a duration Δ_{rev}); upon revocation the policy layer takes actions (safe-stop, notify operator, request re-certification); see Sec. 3.4 for thresholds and hysteresis.

3.3 Formal Gate: PDC & SMT encoding

The Formal Gate implements a design-time EDF feasibility checker for a finite job set $J = \{(r_i, C_i, d_i)\}$.

The gate takes WCET estimates C_i as given. These may come from static-analysis tools such as aiT, from observed execution time maxima plus a safety buffer, or from a hybrid of both. No single method eliminates uncertainty ROS2 middleware jitter and OS scheduling are unavoidable. The certificate's `cert_quality` tag (`exact` `best_effort` records which estimation method was used, and runtime monitoring (Section 3.4) flags cases where observed times consistently exceed certified bounds.

- **Window generation (PDC).** Generate candidate Processor Demand Criterion windows $[t_1, t_2]$ and prune dominated intervals (QPA-style heuristics).
- **SMT encoding (QF_LRA).** Encode each retained window's demand constraint $\sum_{i:\, t_1 < d_i \le t_2} C_i \le t_2 - t_1$ as a QF_LRA query and dispatch the conjunction to Z3 [10].
- **Result handling.** If satisfiable, extract a model to produce the certificate (finish times, per-job min_slack). If UNSAT, obtain an UNSAT core and map it back to jobs/windows to generate actionable repair hints (minimal deadline relaxations or targeted WCET adjustments).
- **Practical mitigations.** To keep the gate tractable, we prune trivial windows, use a configurable SMT timeout and label certificates as `exact` or `best_effort`, and fall back to conservative simulation when the solver times out.

(Full SMT encodings and example certificate JSONs are provided in the anonymised artefact.)

Dedicated algorithms such as QPA [16] solve the same feasibility problem in $O(n^2)$ time without producing diagnostic information. Our SMT encoding is slower but returns an UNSAT core when the instance is infeasible, the basis for the repair hints in Section 4. For typical waypoint counts (3-10 legs), Z3 terminates in under 100 ms. Scaling to longer missions could combine QPA as a fast pre-filter with SMT invoked only on negative instances, although we have not yet implemented this hybrid.

3.4 Runtime Monitoring & Policy

The runtime subsystem produces auditable *evidence tuples* and applies a concise policy mapping evidence to actions (Accept / Degrade / Revoke / Request re-certification).

Monitor duties. A lightweight ROS2 monitor implements:

- **Trace mapping:** observe runtime events (waypoint reached, poses, velocity commands, perception flags) and map them to PTA-labelled transitions using the certificate's event mapping.
- **STL robustness:** compute online STL robustness values with sliding-window algorithms for a small set of timing/safety properties [3, 9, 11].
- **Probabilistic counters:** maintain simple empirical counters for stochastic claims (perception-fail / total-observations) and compute Wilson confidence intervals to compare against certified probabilistic bounds [15].

Evidence tuple. Each monitor report is a compact evidence tuple:

$$E = \langle t,\ property\_id,\ robustness,\ counters,\ pta\_hash,\ diag \rangle,$$

where `robustness` is the STL robustness scalar (or small vector), `counters` records relevant counts, `pta_hash` binds the tuple to a certificate, and `diag` contains optional diagnostics (window id, sensor quality). Tuples are appended atomically to an append-only JSON-lines store and can be packaged with execution traces for offline re-certification and auditing.

Policy mapping. The policy layer consumes certificates and evidence tuples and implements the following rules:

- **Accept:** if all monitored robustness values and counters lie within certified bounds $\Rightarrow$ continue normal operation.
- **Degrade:** if some robustness slack is reduced but safety margins are still acceptable ($\rho(\varphi) < \alpha \cdot \text{min\_slack}$) $\Rightarrow$ switch to conservative mode (reduce speed, tighten sensor fusion).
- **Revoke:** if counters or robustness exceed certified bounds with statistical significance (Wilson interval) and persist for thresholded hysteresis $\Rightarrow$ safe-stop, notify operator, publish revocation record.
- **Request re-certification:** collect trace + counters and submit to offline gate for updated WCET/probabilistic estimates or certificate re-issuance.

Table 1: Summary of illustrative cases (compact).

Case	# Waypoints	Outcome	Key metric / note
A	3	PASS	min_slack = 2.4 s
B	4	UNSAT	suggested δ_{repair} = 1.2 s (deadline extension)
C	3	REVOKE	observed p_{fail} = 0.08 (cert. bound 0.05)

Defaults for practical reproducibility are given in Table 1 (robustness factor α, confidence level for Wilson intervals, hysteresis k, debounce window Δ, SMT timeout) [4, 17]. Transient sensor glitches should not revoke a certificate. The monitor therefore requires k consecutive failures or a sustained violation lasting Δ_{rev} seconds before reporting a revocation. Wilson confidence intervals, which are conservative at low sample counts, further reduce false alarms during mission startup when few observations are available.

Implementation notes (brief). Evidence tuples are stored as JSON-lines (or lightweight append-only DB); each tuple includes the certificate binding (`pta_hash`) and a configuration checksum to detect code/config drift. On detection of drift or repeated violations, the monitor flags the certificate stale, and the policy may trigger re-certification or revocation. Full implementation details, JSON schemas, and packaging scripts are provided in the anonymised artefact.

4 Preliminary Validation: Three Illustrative Cases

While full systematic evaluation is future work, these three cases demonstrate the pipeline's key capabilities and the practical utility of certificate-based assurance. We implemented an anonymised prototype (Extractor, Admission Gate, Runtime Monitor) and an artefact bundle (HotCRP supplemental) with runnable scripts, `pdc.smt2`, and example traces.

Table 1 summarises three compact illustrative runs (PASS / UNSAT / REVOKE); Full traces, SMT logs, and repair-hint details are included in the anonymised artefact.

Case A — PASS. A 3-waypoint mission with conservative WCETs where the Admission Gate returns `Feasible` with min_slack = 2.4s. Runtime evidence tuples confirm robustness margins, and the certificate remains valid for the whole run.

Case B — UNSAT → repair (concrete). Consider a candidate PDC window $[t_1, t_2]$ where the processor demand from the contributing jobs W equals $\sum_{i \in W} C_i = 12.4$ s while the interval length is $L = t_2 - t_1 = 10.0$ s. The conjunction of PDC constraints is therefore

UNSAT. We extract an UNSAT core from Z3 to identify the subset of windows and jobs responsible for the infeasibility [10].

To produce an actionable repair hint, we map the UNSAT core to per-job slack deficits and select a conservative single-step repair: relax the deadline of the job $j \in W$ that has the minimum certified slack (i.e., the job most critically implicated by the core) by

$$\delta \;=\; \sum_{i \in W} C_i - L \;=\; 2.4\ \text{s}.$$

Applying this deadline relaxation and re-dispatching the SMT check in the prototype produced a feasible encoding and yielded a certificate with min_slack = 0.6 s.

We emphasise two practical points. First, the core-to-repair mapping is heuristic: alternative strategies (distributing δ across multiple deadlines or reducing selected C_i estimates) are possible and are explored by the helper script included in the artefact. Second, every suggested repair is re-checked by the Formal Gate (SMT); if the SMT call remains UNSAT, the prototype iterates with alternative repair candidates (or falls back to a best-effort simulation mode). Our approach builds on SMT-based scheduling encodings and practical solver-guided heuristics [13, 14] and leverages Z3's UNSAT-core support [10] to automate the diagnosis-to-repair loop. The artefact includes the `pdc.smt2` instance, the extracted UNSAT core, and a core-to-repair helper script that automates this mapping and re-check loop. The loop terminates when either a feasible schedule is found, the set of candidate repairs is exhausted, or an iteration limit (default 5) is reached; in the latter two cases, the gate falls back to `best_effort` mode.

Case C — REVOKE (at runtime). A mission initially certified as feasible but experiencing perception drift: empirical counters show perception-fail rate exceeding the certified probabilistic bound (Wilson upper bound > certified limit) sustained for k samples. The monitor issues a `Revoke` and the robot safely transitions to a conservative mode; the trace and counters are packaged for offline re-certification.

5 Open Research Challenges

We refine the three research directions defined in Section 1 and now propose the following three directions seem most pressing.

Q1 - Certificate scope Which configuration changes invalidate a certificate, and which do not? Tuning a speed parameter presumably preserves validity; replacing a perception node may not. Formalising this boundary, perhaps via a dependency analysis on the PTA structure, would let operators know when re-certification is required without re-running the full pipeline.

Q2 - Multi-component composition A natural next step is to certify subsystems independently and compose their guarantees. Assume guarantee reasoning for probabilistic timed systems exists [7], but connecting such rules to our certificate schema and runtime evidence remains open. The challenge is ensuring that local certificate validity implies system-level safety when components interact through shared resources.

Q3 - Integration with learned components Neural perception and planning modules have data-dependent latencies that static WCET tools cannot bound tightly. Probabilistic execution-time models or online calibration may be necessary; how these interact with the certificate validity predicate, especially the Wilson interval mechanism for probabilistic claims, is unclear and warrants investigation.

5.1 Related work

Prior work touches parts of our pipeline but does not combine them in the same configuration-bound, machine-actionable way. We briefly position our proposal with respect to four relevant streams.

- **Offline timed / probabilistic verification.** Tools such as UPPAAL and PRISM offer rich formalisms for timing and stochastic analysis, but their artefacts are typically heavyweight models or proofs that are not directly bound to an executable ROS2 configuration. [2, 6]
- **Runtime verification and STL monitoring.** Online monitors provide effective robustness metrics over traces and low-latency detection, yet they do not by themselves supply prior, machine-checkable guarantees that can be consumed as deployment contracts. [3, 9, 11]
- **Safety cases and certification efforts.** Standards and assurance artefacts (e.g., DO-178C) and verified-kernel / verified-compiler projects (seL4, CompCert) emphasise traceable evidence and auditability for certification, but are primarily human-oriented and do not target lightweight, revocable, runtime-checked certificates tailored to robotic missions.
- **Contract and compositional verification.** Assume– guarantee and interface-automata techniques give a foundation for modular reasoning; extending such approaches to probabilistic timed fragments and to certificates that bind assumptions to runtime evidence is an open direction. [5, 7, 12]

In short, our work synthesises elements from these streams: we reuse formal analyses (UPPAAL/PRISM) and monitoring technology (STL) but focus on producing compact, configuration-bound certificates and lightweight runtime evidence pipelines that make offline claims operationally auditable and actionable.

6 Conclusion

We introduced VeriROS and the admission-certificate abstraction: compact, machine-checkable artefacts that bind PTA, verified slack, and probabilistic bounds to concrete ROS2 configurations. Certificates enable pre-launch feasibility gates and runtime evidence-driven accept/degrade/revoke policies, with UNSAT-core-derived repair hints to aid remediation. We provide an anonymised prototype and artefact bundle to stimulate reproducible exploration and invite the community to collaborate on certificate semantics, compositional PTA contracts, and scalable hybrid verification approaches. We view this work as a first step toward principled certificate-driven assurance for autonomous systems, with opportunities for extensions to multi-robot systems, adaptive WCET refinement, and integration with industry safety standards.

Acknowledgments

This work was supported by the John & Pat Hume Doctoral Awards, Maynooth University.

References

[1] Sanjoy K Baruah, Aloysius K Mok, and Louis E Rosier. 1990. Preemptively scheduling hard-real-time sporadic tasks on one processor. In *[1990] Proceedings 11th Real-Time Systems Symposium*. IEEE, 182–190.

[2] Gerd Behrmann, Alexandre David, and Kim Guldstrand Larsen. 2004. A Tutorial on UPPAAL. In *Formal Methods for the Design of Real-Time Systems*. Springer, 200–236.

[3] Alexandre Donzé, Thomas Ferrere, and Oded Maler. 2013. Efficient Robust Monitoring for STL. In *The 25th International Conference on Computer Aided Verification (CAV)*. Springer, 264–279.

[4] Michael Fisher, Emily Collins, Louise Dennis, Matt Luckcuck, Matt Webster, Mike Jump, Vincent Page, Charles Patchett, Fateme Dinmohammadi, David Flynn, et al. 2018. Verifiable self-certifying autonomous systems. In *2018 IEEE international symposium on software reliability engineering workshops (ISSREW)*. IEEE, 341–348.

[5] Arnd Hartmanns and Holger Hermanns. 2009. A Modest Approach to Checking Probabilistic Timed Automata. In *The 6th International Conference on the Quantitative Evaluation of Systems*. IEEE, 187–196.

[6] Marta Kwiatkowska, Gethin Norman, and David Parker. 2011. PRISM4.0: Verification of Probabilistic Real-Time Systems. In *The 23rd International Conference on Computer Aided Verification (CAV)*. Springer, 585–591.

[7] Marta Kwiatkowska, Gethin Norman, David Parker, and Hongyang Qu. 2010. Assume-guarantee verification for probabilistic systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer, 23–37.

[8] Chung Laung Liu and James W Layland. 1973. Scheduling algorithms for multiprogramming in a hard-real-time environment. *Journal of the ACM (JACM)* 20, 1 (1973), 46–61.

[9] Oded Maler and Dejan Nickovic. 2004. Monitoring temporal properties of continuous signals. In *International symposium on formal techniques in real-time and fault-tolerant systems*. Springer, 152–166.

[10] Leonardo De Moura and Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *The 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. Springer, 337–340.

[11] Dejan Nickovic and Oded Maler. 2007. AMT: A Property-Based Monitoring Tool for Analog Systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*. Springer, 304–319.

[12] Gethin Norman, David Parker, and Jeremy Sproston. 2013. Model checking for probabilistic timed automata. *Formal methods in system design* 43, 2 (2013), 164–190.

[13] Victoria Marie Tuck, Pei-Wei Chen, Georgios Fainekos, Bardh Hoxha, Hideki Okamoto, S. Shankar Sastry, and Sanjit A. Seshia. 2024. SMT-Based Dynamic Multi-Robot Task Allocation. In *The 16th International Symposium on NASA Formal Methods (NFM)*. Springer, 331–351.

[14] Shimin Wang, Xiaojuan Liao, Min Wang, Luyue Chang, Huan Yang, and Tao Wang. 2020. An Improved SMT-Based Scheduling for Overloaded Real-Time Systems. *Engineering Letters* 28, 1 (2020).

[15] Edwin Bidwell Wilson. 1927. Probable Inference, the Law of Succession, and Statistical Inference. *J. Amer. Statist. Assoc.* 22, 158 (1927), 209–212.

[16] Fengxiang Zhang and Alan Burns. 2009. Improvement to quick processor-demand analysis for EDF-scheduled real-time systems. In *2009 21st Euromicro Conference on Real-Time Systems*. IEEE, 76–86.

[17] Tong Zhao, Ekim Yurtsever, Joel A Paulson, and Giorgio Rizzoni. 2022. Formal certification methods for automated vehicle safety assessment. *IEEE Transactions on Intelligent Vehicles* 8, 1 (2022), 232–249.