

A declarative and probabilistic model with rewards to formally verify satellite missions [★]

Fernando Asteasuain^{1,2}, Danilo Medina¹, and Ana Legaspi¹

¹ Universidad Nacional de Avellaneda, Argentina
`fasteasuain@undav.edu.ar`

² Universidad Abierta Interamericana - Centro de Altos Estudios
CAETI, Argentina

Abstract. Software Probabilistic models can be extended with the concept of rewards to enrich and enhance the behavioral properties to be specified and studied in a software system. In the last years, rewarded probabilistic models have been employed to formally verify cutting-edge domains such as satellite missions specifications. In this work we present an extension of our flexible and graphical specification framework called FVS incorporating rewards into our probabilistic model. We validated our approach by modeling a real and complex case of study for a nano-satellite and we also developed an important software tool to simulate satellite missions and recollect data.

Keywords: Formal Verification, Probabilistic Models, Satellite Missions

1 Introduction

Probabilistic models have always been a resourceful asset to consider in Software Engineering. In the last years they have been effectively applied in groundbreaking software domains such as satellite mission or robotic systems specification [23, 26]. In particular, when applied to formal verification phase probabilities show three main flavours. The first one is to tackle a widely known problem in formal verification which is the state-explosion problem [30]: the model to be explored is early too soon too big and the automatic tools such as model checkers are not able to verify a given property in a reasonable amount of time. Adding probabilities into the model reduces the space to be explored with the cost of losing some precision. For example, instead of saying the system will always satisfy a given property, the model checker can say that the property will be satisfied with a 0.98 probability. The second flavour is combined with the concept of software uncertainty [31, 17, 24], that is, being able to cope with requirements that are suitable to change or that remains unclear in early stages. For example, a possible way to model uncertainty in Self-Adaptive Systems (SAS) [24, 17] is introducing probabilities into the model [7, 27]. This enables the possibility to express that a certain behavior is likely to occur with a given probability. Finally, the third flavour of probabilistic software engineering might be its usage

[★] This work was partially funded by UNDAVCYT 2025

on formally verifying large systems and large models [21, 28]. In this case, like in the state-explosion problem, probabilities help to diminish the space to be explored and analyzed.

Probabilistic models can be easily and straightforwardly extended with the concept of rewards [1, 10, 19]. Rewards can be viewed as labels in a Labelled Transition System (LTS) [29]: an easy way to enrich a model to provide more computational power. Rewards can also be viewed as a way to ponder or weight some decisions in the software to be analyzed [1, 10]. For example, rewards can be interpreted as costs to indicate that certain operations or transitions of the system are more “expensive” than others. These values can be accumulated and the software engineer could express properties regarding that accumulated value to introduce thresholds, a typical software requirement to be addressed. In LTS, rewards can be incorporated when entering or exiting a given system state or directly associated to a transition relating two states [1, 10, 19].

Several approaches incorporated rewards to their probabilistic model [1, 19, 9, 16]. However, few of them allow to denote behavior in a declarative way, closer to the way requirements are expressed. Being able to express behavior declaratively is always a crucial aspect when specifying behavior [20]. In this sense, in [13, 14] the probabilistic version of the FVS (Feather Weight Visual Scenarios) specification framework was introduced. FVS [2, 4, 5] is a simple yet powerful and flexible graphical specification language. In previous work probabilistic FVS [13, 14] has been successfully applied in cutting edge domains such as satellite missions and robotic specifications. We now build on the top of these works introducing Rewarded FVS, a simply FVS extension including a probabilistic model with rewards. We validated our approach modeling a real and complex case of study: an specification from a Cube-Sat satellite mission and three particular properties extracted from the literature considered as patterns since the behavior they denote is meaningful and expressive [23, 12, 8]. We also developed an important software tool to simulate satellite missions and recollect enough data and information to provide purposeful values to the rewarded probabilistic model. Formal semantics of Rewarded FVS will be addressed in future work.

The rest of this paper is structured as follows. Section 2 introduces the main concepts of the Rewarded FVS specification language whereas Section 3 shows our language in action considering our Cube-Sat mission specification. Finally, Sections 4 and 5 present related and future work and the final conclusions of this work.

2 Rewarded FVS

We first introduce FVS basic concepts and Section 2.1 describes how rewards are incorporated to our language. The reader is referred to [2] for a formal characterization of the language. FVS is a graphical language based on scenarios. Scenarios are partial order of events, consisting of points, which are labeled with a logic formula expressing the possible events occurring at that point, and arrows connecting them. An arrow between two points of interest represent a

total function between points establishing a **precedence relationship** between them. An arrow connecting a point A to a point B indicates that the *A* event occurs before the occurrence of point *B*. FVS also features a function to restrict behavior between points, which is called the **forbidden function**. This function allows to express that certain behavior must not occur between points, which is a classical requirement for behavioral specification languages. The forbidden relationship is graphically shown as a label in the arrows connecting points.

We now introduce the concept of FVS rules, a core concept in the language. An FVS rule consists of two or more scenarios: a scenario playing the role of an antecedent and at least one scenario playing the role of a consequent. The intuition is that whenever a trace “matches” a given antecedent scenario, then it must also match at least one of the consequents. In other words, rules take the form of an implication: an antecedent scenario and one or more consequent scenarios.

As fully detailed in [13, 14] probabilities are introduced in FVS as follows. We consider that a system will satisfy a given FVS rule *R* with a probability *pr* as the ratio between the number of possible behaviors satisfying the rule and the total number of possible behaviors. In raw words, we say the rule *R* is satisfied in at least *pr* % of the cases (*pr*% stands for $100 * pr$ %). This is indicated by introducing a mathematical function *f* which assigns probabilities to FVS behavioral scenarios.

A complete probabilistic FVS rule is shown in Figure 1. The rule addresses a requirement from our case of study: a specification for a Cube-Sat satellite mission: when satellite lights are turned on during an eclipse a high level consumption alert must be dispatched with a probability higher than 0.99. The rule consists of an antecedent scenario, where a *EclipseDetected* event is followed by a *LightsOn* event. During the time elapsed between these two events is must not occur the *EclipseEnd* event. These scenarios represent the situation where lights are turned on during an eclipse. The consequent scenario (in grey in Figure 1) establishes that the *HC-AlertDispatched* must occur, checking that the required alert is correctly dispatched, fulfilling the mentioned requirement. Additionally, this must occur with a probability greather than 0.99. As it is shown, probabilities are simply stated as labeled arrows between points (events).

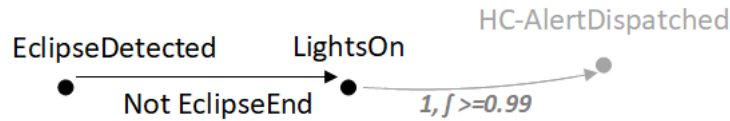


Fig. 1. An FVS rule example with probabilities

2.1 Introducing Rewards: The Rewarded FVS language

Rewards are introduced seamlessly in Probabilistic FVS. In the same way that probabilities were added to the original FVS as a mathematical function between points, rewards are also contemplated as a function we named *kappa* κ to establish a reward (or a cost) between two connected points. For example, in the requirement mentioned earlier we could add a cost to penalize when lights are turned on during an eclipse. This is exhibited in the upper Rewarded-FVS rule in Figure 2 where the κ function is assigned a 0.5 value. Similarly, we introduce a $\kappa\text{-acc}$ function to denote the accumulated values of rewards. Continuing the same example, in the bottom rule Rewarded-FVS rule in Figure 2 it is stated that the accumulated must not exceed a given 0.95 threshold dictated by the satellite specification.

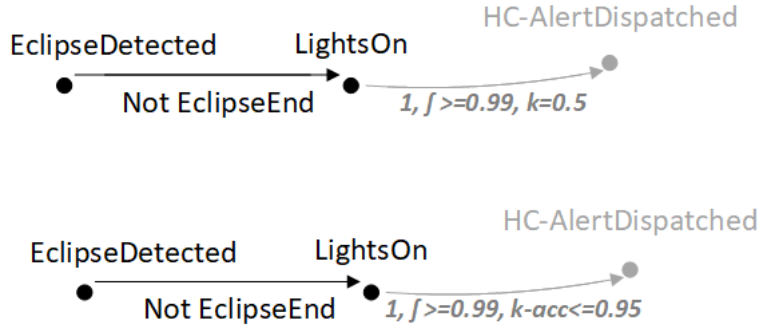


Fig. 2. An FVS rule example with rewards and accumulated rewards

3 Case of Study: Cube-Sat Behavior Specification

Among nano-satellites, the Cube-Sat project has emerged lately as one of the most popular ones [25]. We developed a meaningful case of study consisting in a full document of more than thirty pages describing the expected behavior of a Cube-Sat satellite mission. The requirements were obtained from real Cube-Sat missions taken from the literature and represent real specification assignments [11, 18]. In concrete, we fully studied three substantial behavioral properties considered as patterns since the behavior they denote is meaningful and expressive [23, 12, 8]. We named these properties as: The Stand By Mode, Secular Behavior and the Solar Energy Optimization, which are covered in the next subsections. For their modelling we employed several of the quantitative specification pattern introduced by [23]. Finally, our software simulation tool is presented in Section 3.4.

3.1 Stand By Mode Operational Behavior

An important requirement from the solar panel component is the following: Critical units energy should be minimized when the bus voltage is between 70% and 85% of the optimal values with a probability equal or greater than 0.93. This can be modeled employing the Minimize Quantitative pattern [23]. This pattern guarantees that, with a given probability, the system will (minimize) a given quantitative measure t while performing a mission. This behavior could also be viewed as an instantiation of the *Stand by Mode* introduced in [12]. In this mode only those components which are necessary to monitor the system and to survive the external environment shall be active. The Rewarded Probabilistic FVS specification for this behavior is captured Figure 3, where we assumed a 0.91 value for the maximum threshold for the bus voltage. The CU acronym used in the events in this text stands for Critical Units. The *kappa* function, employed as an accumulative rewards interpreted as costs, controls the critical units energy.

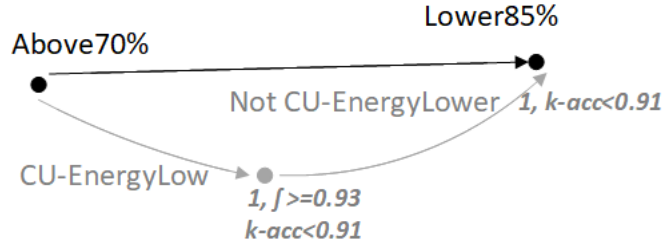


Fig. 3. Stand by Mode operation with rewards

The *kappa* function can also penalize or rewards certain satellite behavior regarding critical units energy as it is shown in Figure 4. In this case, rewards as costs are used to penalize turning on the satellite lighting system during an Eclipse since during an eclipse satellite's resources' consumption are higher. This is achieved by assigning a much lower cost when lights are turned in a normal mode than when is done during an eclipse (0.1 against 0.5).

3.2 Satellite's Secular Behavior

In space and physics, secular refers to long-term, non-periodic changes or trends that happen gradually over years or centuries. In a satellite mission specifications it is important to measure accumulative, long-term orbital changes caused by Earth's oblateness, atmospheric drag, or the gravitational pull of the Sun and Moon [22]. This can be related to satellite's battery voltage preservation under some satellite maneuvers and the Preservation Quantitative Pattern introduced in [23]. Some approaches pinpoint the importance of measuring values like power

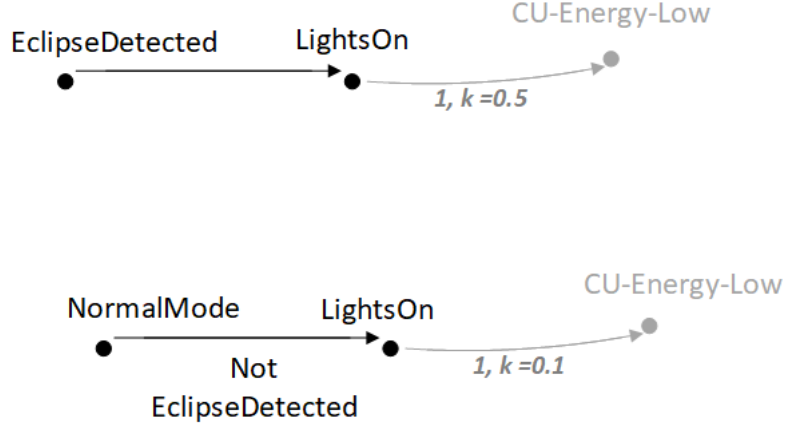


Fig. 4. Rewards as costs involving resources' consumption during an eclipse

voltage in satellite mission specification [15, 8] since it can be underestimated in simulated environments [8]. As an example consider the following requirement for our case of study: The Telemetry, Tracking and Command (TCC) system shall continue to operate functionally down to a voltage 80 % worst- case bus voltage with a probability equal or greater than 0.99. This requirement could be specified using Rewarded FVS instantiating the Preservation Pattern. Rewards can be used to penalize voltage operations, and accumulative values can delimit satellites behavior establishing some maximal or minimal accepted thresholds as illustrated in Figure 5 employing the *kappa-acc function*.

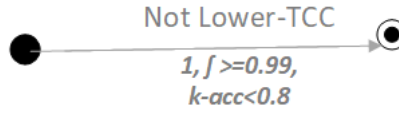


Fig. 5. Voltage Preservation modeled with accumulative rewards

In the following Rewarded FVS scenarios (Figure 6) two critical satellite operations are assigned different costs (0.66 and 0.12) reflecting how they affect the voltage usage or consumptions. In this case, the tracking operation is 0.66 whereas the curvature operation is assigned 0.12, following the satellite mission specification of our case of study.

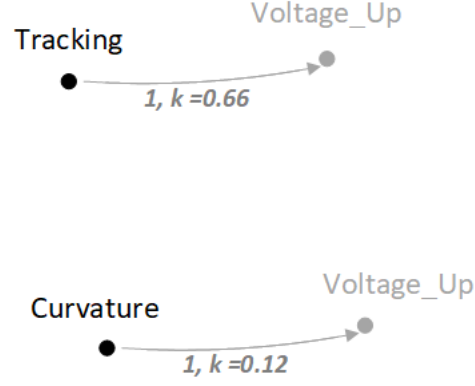


Fig. 6. Voltage Preservation specified with rewards viewed as costs

3.3 Solar Energy Optimization

We employed the Maximize Quantitative Pattern [23] regarding solar energy optimization for satellite's solar panels. One of the requirements of our case of study can be expressed as follows: The power bus shall be regulated during sunlight operation. During eclipse, the bus voltage shall be maximized with probability equal or greater than 0.97. In this case, assuming a threshold of 0.87 we could employ accumulative rewards as illustrated in Figure 7.

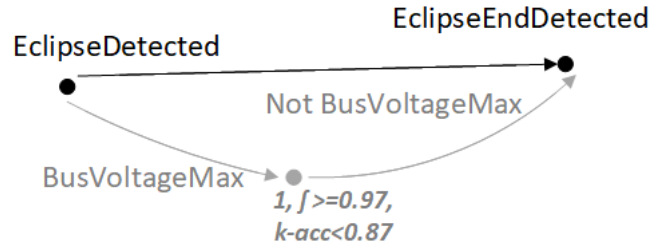


Fig. 7. Solar Energy Optimization and accumulative rewards

Similarly to the specification for the secular orbit in Section 3.2 rewards can be interpreted as costs to penalize operations that affect the solar energy consumption modeled in Figure 7. For example, the Solar Panel Opening event is assigned 0.11 whereas the Solar Panel Maintaining event is assigned 0.55 due to its high bus voltage consumption. This is illustrated in Figure 8.

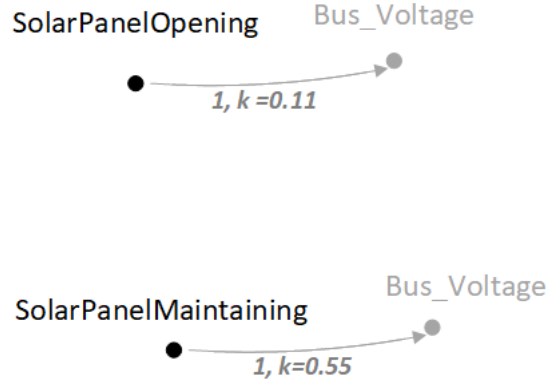


Fig. 8. Rewards in the Solar Panel Operation

3.4 Software Simulation Framework for Satellite Missions

We developed a software framework to customize software satellite missions which are later on simulated on the NASA simulation tool General Mission Analysis Tool (GMAT) ³. Our software framework is based on Python scripts and it is available on a GitHub repository ⁴. This tool allowed us to connect directly with the NASA tool accessing by console commands. We build a setup for a 3kg nano-satellite (an standard Cube-Sat 3U) in an sun-synchronous orbit with a semi-major axis of 7178.136 km and an inclination of 98 degrees. This simulation software framework allowed us to recollect precise data to provide accurate values to our probabilistic model with rewards for our case of study.

4 Related and Future Work

The most imminent research line continuing this work is to formally introduce Rewarded FVS. In other aspects, several approaches includes probabilistic models with rewards to formally verify software [1, 19, 9, 16]. The Storm model checker [16] is an appealing software tool that supports the analysis of discrete- and continuous-time variants of both Markov chains and Markov decision processes. Storm handles several input specification languages such as generalized stochastic Petri nets. In this sense, it would be interesting to explore to employ FVS as input to the Storm model checker. Rewards are also first-class citizens in probabilistic models such as the ones presented in [1, 19]. Finally, [9] introduces a powerful model checker called ULTIMATE, which stands for Universal stochasTic Modelling, verificAtion and synThEsis framework. Besides providing

³ <https://etd.gsfc.nasa.gov/capabilities/capabilities-listing/general-mission-analysis-tool-gmat/>

⁴ <https://github.com/danilomedina12/Nanosatellite-ejecutor-simulaciones-gmat>

probabilistic verification, ULTIMATE handles behavioral synthesis [6], which is an efficient way of dealing with requirements based on theory-games algorithms. FVS can also handle behavioral synthesis, [3–5] so a reasonable future line of work to explore is the possibility of combining these two approaches.

5 Conclusions

In this paper we extended our probabilistic specification framework FVS with rewards, a useful mechanism in probabilistic models. We applied rewards in a complex and challenging case of study involving a cutting edge domain such as satellite missions specifications. We also developed a software framework tool to customize and simulate satellite missions. Rewarded FVS proved to be in the right path to become an alluring framework to formally specify, verify and synthesize behavior for software satellite-based systems.

References

1. S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model-checked. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 88–104. Springer, 2003.
2. F. Asteasuain and V. Braberman. Declaratively building behavior by means of scenario clauses. *Requirements Engineering*, pages Vol 22,239–274, 2017.
3. F. Asteasuain and V. Braberman. Declaratively building behavior by means of scenario clauses. *Requirements Engineering*, 22(2):239–274, 2017.
4. F. Asteasuain and L. R. Caldeira. A sound and correct formalism to specify, verify and synthesize behavior in big data systems. In *Argentine Congress of Computer Science*, pages 109–123. Springer, 2022.
5. F. Asteasuain, F. Calonge, M. Dubinsky, and P. Gamboa. Open and branching behavioral synthesis with scenario clauses. *CLEI E-JOURNAL*, 24(3), 2021.
6. R. Bloem, B. Jobstmann, N. Piterman, A. Pnueli, and Y. Sa’Ar. Synthesis of reactive (1) designs. 2011.
7. A. Cailliau and A. V. Lamsweerde. Runtime monitoring and resolution of probabilistic obstacles to system goals. *ACM Transactions on Autonomous and Adaptive Systems (TAAS)*, 14(1):1–40, 2019.
8. R. Caldas, J. A. P. García, M. Schiopu, P. Pelliccione, G. Rodrigues, and T. Berger. Runtime verification and field-based testing for ros-based robotic systems. *IEEE Transactions on Software Engineering*, 50(10):2544–2567, 2024.
9. R. Calinescu, S. Gerasimou, S. Getir Yaman, G. VAZQUEZ FLORES, and M. Bassett. Verification of multi-model stochastic systems. In *ICSE 2026*. IEEE, 2026.
10. R. Calinescu, C. Ghezzi, K. Johnson, M. Pezzé, Y. Rafiq, and G. Tamburrelli. Formal verification with confidence intervals to establish quality of service properties of software systems. *IEEE transactions on reliability*, 65(1):107–125, 2015.
11. M. Doyle, R. Dunwoody, G. Finneran, D. Murphy, J. Reilly, J. Thompson, S. Walsh, J. Erkal, G. Fontanesi, J. Mangan, et al. Mission testing for improved reliability of cubesats. In *International Conference on Space Optics*, volume 11852, pages 2699–2718. SPIE, 2021.
12. M. Etumi, H. M. Taylor, and M. Farrell. Towards a catalogue of requirement patterns for space robotic missions. *arXiv preprint arXiv:2511.14438*, 2025.

13. F.Asteasuain, A.Legaspi, G.Freire, and M.Machuca. A declarative and probabilistic behavioral specification language to handle uncertainty in modern software development. In *CONAIISI*, 2024.
14. F.Asteasuain, A.Legaspi, J.Acuna, M.Machuca, and G.Freire. A declarative and flexible specification for probabilistic and quantitative behavioral patterns. In *CONAIISI*, 2025.
15. M. I. Gallardo Duval. Optimizing satellite operations: uncertainty modeling and integration in constellation scheduling. Master Thesis. UPC. 2024.
16. C. Hensel, S. Junges, J.-P. Katoen, T. Quatmann, and M. Volk. The probabilistic model checker storm. *International Journal on Software Tools for Technology Transfer*, 24(4):589–610, 2022.
17. C. Krupitzer, F. M. Roth, S. VanSyckel, G. Schiele, and C. Becker. A survey on engineering approaches for self-adaptive systems. *Pervasive and Mobile Computing*, 17:184–206, 2015.
18. E. Kulu. Nanosatellite and cubesat database, <https://www.nanosats.eu/>.
19. M. Kwiatkowska, G. Norman, and D. Parker. Prism 4.0: Verification of probabilistic real-time systems. In *CAV*, 585–591.
20. A. V. Lamsweerde. Goal-oriented requirements engineering: A guided tour. In *RE*, 2001.
21. C. P. Lee, D. Porfirio, X. J. Wang, K. C. Zhao, and B. Mutlu. Veriplan: Integrating formal verification and llms into end-user planning. In *Proceedings of the 2025 CHI Conference on Human Factors in Computing Systems*, pages 1–19, 2025.
22. E. Legnaro and C. Efthymiopoulos. Secular dynamics and the lifetimes of lunar artificial satellites under natural force-driven orbital evolution. *Acta Astronautica*, 225:768–787, 2024.
23. C. Menghi, C. Tsigkanos, M. Askarpour, P. Pelliccione, G. Vázquez, R. Calinescu, and S. García. Mission specification patterns for mobile robots: Providing support for quantitative properties. *TSE*, 49(4):2741–2760, 2022.
24. I. Pekaric, R. Groner, T. Witte, J. G. Adigun, A. Raschke, M. Felderer, and M. Tichy. A systematic review on security and safety of self-adaptive systems. *Journal of Systems and Software*, 203:111716, 2023.
25. N. Saeed, A. Elzanaty, H. Almorad, H. Dahrouj, T. Y. Al-Naffouri, and M.-S. Alouini. Cubesat communications: Recent advances and future challenges. *IEEE Communications Surveys & Tutorials*, 22(3):1839–1862, 2020.
26. A. Singla, I. Gallardo-Duval, A. Calveras, J. A. Fraire, and J. A. Ruiz-De-Azua. Probabilistic-aware satellite constellation scheduling for integrated tn-ntn operations. *IEEE Open Journal of the Communications Society*, 2025.
27. G. F. Solano, R. D. Caldas, G. N. Rodrigues, T. Vogel, and P. Pelliccione. Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach. In *2019 IEEE/ACM SEAM*, pages 89–99. IEEE, 2019.
28. N. Tihanyi, T. Bisztray, M. A. Ferrag, B. Cherif, R. A. Dubniczky, R. Jain, and L. C. Cordeiro. Vulnerability detection: from formal verification to large language models and hybrid approaches: a comprehensive overview. *Adversarial Example Detection and Mitigation Using Machine Learning*, pages 33–47, 2026.
29. J. Tretmans. Model based testing with labelled transition systems. In *Formal methods and testing, pages=1–38, year=2008, publisher=Springer*.
30. A. Valmari. The state explosion problem. In *Advanced Course on Petri Nets*, pages 429–528. Springer, 1996.
31. T. Vogel, M. Carwehl, G. N. Rodrigues, and L. Grunske. A property specification pattern catalog for real-time system verification with uppaal. *Information and Software Technology*, 154:107100, 2023.