A Declarative and Flexible Specification for Probabilistic and Quantitative Behavioral Patterns

Fernando Asteasuain [1], Ana Legaspi [1],Jorge Acuña[1], Martin Machuca[1],Gastón Freire[1]
{fasteasuain, alegaspi,jacunna,mmmachuca,gafreire@undav.edu.ar}
[1] Universidad Nacional de Avellaneda. Dpto. de Tecnología y Administración. Ingeniería en Informática

Abstract

Probabilities and quantative measures represent a key feature in modern software development which is being applied in diverse domains such as robotic and satellite missions or Self-Adaptive Systems. In this work we present a declarative and graphical framework called Probabilistic FVS (Feather Weigh Visual Scenarios) to specify systems' behavior with probabilistic and quantative capabilities. This framework is expressive enough to model most of the most known probabilistic specification patterns. Using patterns is a neat improvement to ease the task of specifying the expected behavior of systems and formally verified them with tools like model checkers. Our approach is validated employing a real case of study base on a nano-satellite mission. We also developed a user-profile interaction to obtain meaningful probabilistic measures for the case of study.

Keywords: Formal Verification, Probabilistic Models.

1. Introduction

Probabilistic models have always been present in software development [1-5]. One of the most puissant reasons for this is that software engineers are rarely in full control of how the system will flow and execute. Software Engineering, a dominant domain in Computer Science, has been proposing methodologies, tools and techniques to mitigate this issue. For example, the use of software analysis and testing are key aspects to assure software quality [6-7].

However, when due to resources limitations or practical reasons (for example, a large and distributed system which is difficult to analyze as a whole) the possibility to asset with accuracy and precision if a system behaves as expected probabilities become a handy tool to provide satisfactory answers.

Probabilities may appear in software development under several costumes. For example, is a common weapon to tackle the concept of uncertainty in software systems that must adapt quickly for changes in the environment where they are deployed and executed. This is the case Self-Adaptive Systems (SAS) [8-10]. SAS employ a layered architecture and a loop-based cycle where changes are detected, modifications applied and a new configuration is launched [10-13]. SAS have been successfully applied in a plethora of pioneering domains such as UAV's (Unmanned Aerial Vehicles), nano satellites, multiagent systems, fault-tolerant computing, dependable computing, distributed systems, autonomous computing, self-managing systems, autonomous communications, adaptable user interfaces, machine learning, economic and financial systems [10-18]. A possible way to model uncertainty in SAS is introducing probabilities into the model [16-18]. This enables the possibility to express that a certain behavior is likely to occur with a given probability.

Probabilities are also used in model checking tools [3-4,17,37,52,53]. A model checker is a formal verification tool that given a property describing an excepted behavior of the system and model describing how the system works it says if the model satisfies the desired property or not. In the latter case, the model checker provides a counter example that leads to the violation of the property. Although the meaningful contributions of model checking sometimes the amount of space to be checked is too vast and no answer is obtained from the tool in a reasonable time. A probabilistic model checker is a viable option for these cases. It cannot guarantee that a property will always be satisfied but it could establish that it will be satisfied in 99 out of 100 possible executions, which might be reasonable enough for several domains and systems.

The behavioral modeling of certain type of requirements withholds a probabilistic and quantitative nature which is an easier to capture with probabilistic mechanisms. The so called "nonfunctional" requirements such as Middle Time to FAILURE (MTTF), probability of failure upon demand (PFD), availability, performance or reliability fall in this category. Some classical examples of these requirements are: lower or uppers bounds on the time a system take to perform some action, energy consumptions requirements for a satellite or a robot battery [17-22].

Several formalisms such as Temporal Logics have been extended to express probabilities. For example, Probabilistic reward computation tree logic (PRCTL) PRCTL properties are interpreted over discrete-time Markov reward models (e.g., [23]), i.e., state machines containing states labelled with probabilities and rewards.

To simplify writing behavior in these formalisms traditional patterns specification [24] were also extended to denote probabilities [21-22]. Work in [24] introduces a set of templates capturing recurring solutions for the behavioral specification phase: the specification patterns. The most widely known example is the *Response pattern*, which essentially says that every stimulus must be followed by a response, a classical requirement in responsive systems. Work in [22] extends these patterns to include probabilities while work in [21] introduces patterns with a quantative vision of requirements being especially suited for domains such as robotic or satellite missions.

In this sense, it is also important to express requirements and behavior in a declarative way. A declarative specification formally denotes requirements closer to the way they are stated in natural language. This feature eases the process of introducing changes and general analysis as well as behavioral exploration compared to other operational notations [23,24].

Given this context in this work we further deepen the usage of probabilities in the flexible, graphical, expressive and declarative language FVS (Feather Weight Visual Scenarios) [25-27]. FVS has been proved more flexible and expressive that other formalisms such as Linear Temporal logics [32]. In [27] a probabilistic version of FVS was introduced.

We now continue this line of research in a twofold effect. First, we modeled in FVS the two most applied set of probabilistic behavioral specification patterns: all the quantative patterns catalog [21] and all the patterns included in the probabilistic extension [22]. This exemplifies the versatility of the FVS notation. Secondly, we applied both the probabilistic specification patterns in a real, concrete and challenging case of study. We developed a meaningful case of study consisting in a full document describing the expected behavior of a CubeSat satellite mission [28]. It is conceived as a general specification template since additional requirements should be added to address a specific mission. It is a very complete document of more than thirty pages specifying the behavior of several satellites' artifacts and component. The case of study is striking since the CubeSat project has emerged lately as one of the nano-satellites [29]. Probabilities were added to our case of study following the technique in [16]. This approach introduces probabilities based on how the user interacts with the system based on user profiles. For building "user-profile interaction" for our case of study we obtained a vast number of values and data from existing and simulated nano satellites missions from several sources.

The main contributions for this paper might be synthesized as:

- A Declarative and flexible specification for probabilistic patterns extending the original specification patterns.
- A Declarative and flexible specification for the complete catalog of quantitative patterns for robotic and satellite missions.
- A validation of FVS probabilistic patterns specification usage in a real, concrete and appealing case of study: a fully document describing requirements for a Cube-Sat nano satellite mission.
- The development of a solid "user-profile" interaction to incorporate realistic probabilities into our model. This challenge included the use and exploration of an important amount of data and analysis from several sources such as satellite simulation software and logs for previous missions.

The rest of the paper is structured as follows. Section 2 Background introduces the main features of FVS and Probabilistic FVS. Section 3 presents how two of the most widely known probabilistic patterns catalog are modeled in Probabilistic FVS. Section 4 introduces our case of study and details how probabilities were obtained whereas Section 5 exhibits how the FVS behavioral patterns specification is applied to the case of study. Section 6 and 7 introduce related and future work. Finally, Section 8 enumerates the final conclusions for this work.

2. Background: FVS main features

In this section we will informally describe the standing features of FVS [25,26]. The reader is referred to [25] for a complete 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 indicates precedence of the source with respect to the destination: for instance, in Figure 1-a A-event precedes B-event. We use an abbreviation for a frequent sub-pattern: a certain point represents the next occurrence of an event after another. The abbreviation is a second (open) arrow near the destination point. For example, in Figure 1-b the scenario captures the very next B-event following an A-event, and not any other B-event. Conversely, to represent the previous occurrence of a (source) event, there is a symmetrical notation: an open arrow near the source extreme. For example, in Figure 1-c the scenario captures the immediate previous occurrence of a B-event from the occurrence of the A-event, and not any other B-event. Events labeling an arrow are interpreted as forbidden events between both points. In Figure 1-d A-event precedes B-event such that C-event does not occur

between them. FVS features aliasing between points. Scenario in 1-e indicates that a point labeled with A is also labeled with A ^ B. It is worth noticing that A-event is repeated on the labeling of the second point just because of FVS formal syntaxes [25]. Finally, two special points are introduced as delimiters to denote the beginning and the end of an execution. These are shown in Figure 1-f.

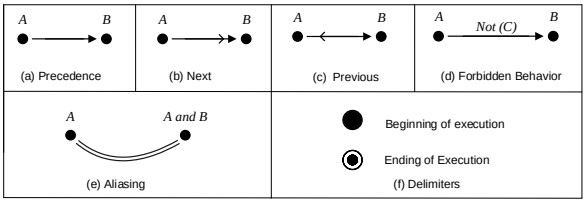

Figure 1. Basic Elements in FVS

We now introduce the concept of FVS rules, a core concept in the language. Roughly speaking, a rule is divided into two parts: a scenario playing the role of an antecedent and at least one scenario playing the role of a consequent. The intuition is that if at least one time the trace "matches" a given antecedent scenario, then it must also match at least one of the consequents. Graphically, the antecedent is shown in black, and consequents in grey. Since a rule can feature more than one consequent, elements which do not belong to the antecedent scenario are numbered to identify the consequent they belong to.

An example is shown in Figure 2 considering a typical requirement for a satellite mission. The requirement being modeled is the following: *The Attitude and Orbit Control Subsystem (AOCS) must monitor the control performance and attitude sensors until the final de-orbiting maneuve*r. To that end, we introduce four events: **InOrbit, DeOrbit, ControlPerformance** and **AttitudeSensorOK**. The first two events address the starting and ending points of the maneuver action where the last two events focus on the parameters controllers for performance and attitude.

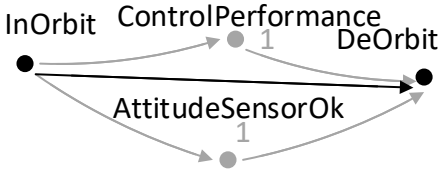

Figure 2. A Rule modeling a requirement for a satellite mission.

The rule in Figure 2 is simply stating that performance and attitude must be monitored while maneuvering. In events terms, **ControlPerformance** and **AttitudeSensorOK** must occur while maneuvering (the timeline between **InOrbit** and **DeOrbit** occurrences).

2.1 Probabilistic FVS

The probabilistic flavor of FVS was introduced in [27] It consists of a simply extension, where a function assigning probabilities is attached to the precedence and forbidden relationships between points. Inspired on the model introduced in [16] we consider a lower bound for an event to occur. That is, we focus on the lowest (or highest) chance of an event occurrence. Regarding FVS rules, we consider that a system will satisfy a given 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 conservative approach is one of the most commonly used in formal verification software since the objective is to be as accurate as possible without introducing false positives [30].

To introduce an example, we might revisit the AOCS subsystem requirement modeled in Figure 2. By analyzing previous satellites missions' data and other sources like software simulation we can say that the attitude sensors and performance controls shall be monitored with *a probability equal or higher than 0.99*. The probabilistic FVS rule is shown in Figure 3. It can be seen graphically that the probabilistic rule in Figure 3 is almost identical to the non-probabilistic rule in Figure 2. The only difference is the addition of the probability, indicated by the formula *∫ >=0.99 attached* to the consequent of the rule, which is the function *∫* that assigns probabilities to each event.

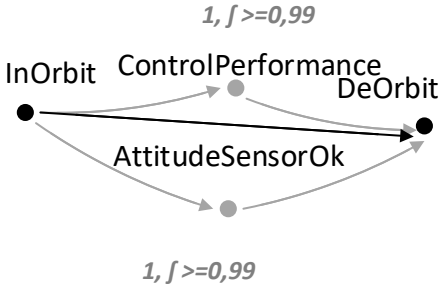

Figure 3. A probabilistic FVS Rule

3. Probabilistic Behavioral Patterns and their codification in Probabilistic FVS

The usage of repeating a formula or technique used successfully in the past to solve a similar actual problem is a very common approach not only in computers science but to everyday life for a very long time in human story. In Computer Science perhaps the most popular appearance was the Design Patterns [31] introduced in the middles 90'. This work captures a set of recurrent and common structures end behavior to solve a particular problem, which appears in similar ways in different problems. Each pattern constitutes a template that can be instantiated in every new domain providing a tested and verified solution employed successfully in the past. In addition, it also

brings a communication identity which is a key feature. For example, a software engineer might say "I applied the Composite pattern to solve the database transaction problem" and everyone can understand with this simple statement the software artifacts involved in the solution and how the interact and communicate.

This solution was translated to the specification world by [24]. Specification patterns were introduced to simplify the usage of complex formal languages such as Linear Temporal Logics (LTL) [32]. For example, one the most known pattern is the Response Pattern. This pattern captures a very common requirement in reactive system: *every stimulus must be followed by a response*. In a client server system this pattern can be instantiated as: *every client request must have a server's response*.

The incorporation of quantative aspects and probabilistic in formal models made that behavioral pattern specification mutated to cope with these issues [21,22]. We can classify these patterns into two categories: those focused on quantitative aspects and those extending the traditional patterns to add a probabilistic flavor. The former category deals with aspects such as maintainability, performance, reliability, maximizing and minimizing measures, etc. The latter category is a more direct approach: they simple extend traditional specification patterns to cope with probabilities.

We have chosen two popular probabilistic patterns in each category: [21] for quantative patterns and [22] for the probabilistic version of traditional patterns. In this way we can exhibit the potential and expressive power of Probabilistic FVS to specify both kinds of probabilistic patterns. The next two subsections are focused on these topics.

3.1 Quantitative Pattern Specification in FVS

In this section we describe how the quantitative patterns proposed in [21] are modeled in FVS. The catalog of quantitative patterns includes behavior for modeling intervals (Within pattern, Strictly Within pattern), for modeling objectives (Maximize and Minimize Pattern), for shaping Bounds (At Least, Exactly, At Most patterns), for modeling Performance and Dependability issues (Confidently or Reliability patterns), for modeling timed requirements (Simultaneously, End, Pause or Timeout patterns) or to model Resources behavior such as the Preservation pattern. The complete catalog is detailed in [21].

The Maximize and Minimize pattern guarantee that, with a given probability, the system will maximize (minimize) a given quantitative measure *t* while performing a mission. This is modeled in FVS as shown in Figure 4. The rule for the Maximize pattern in the top of the figure indicates that no values greater than *t* are found until the end of the execution once value *t* occurs with a probability greater than *pr*. The rule for the Minimize pattern is modeled similarly and it is shown in the lower part of Figure 4. Probabilistic values such as *t* are viewed as events to simplify the specification.

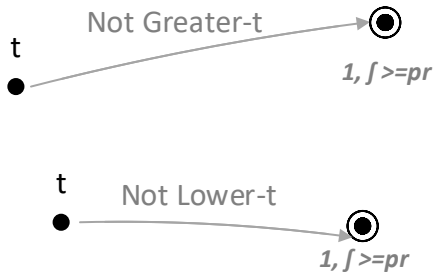

Figure 4. Maximize and Minimize patterns in FVS

The At Most, At Least and the Exactly patterns specify that a certain measure *k* will remain in the desired spectrum (at most/at least or exactly) with a probability greater than *pr*. FVS rules in Figure 5 capture the behavior of these three patterns.

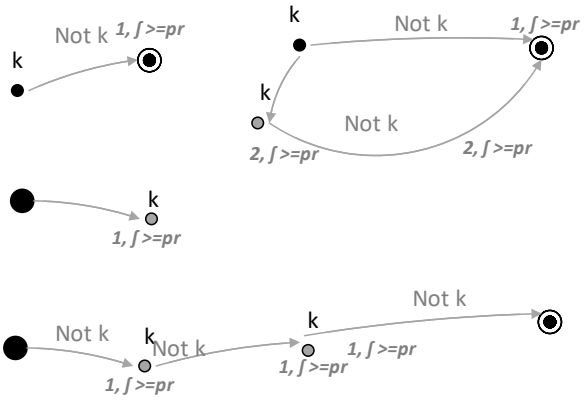

Figure 5. At Most, At Least and Exactly patterns in FVS

The two rules at the top of the Figure capture the At Most pattern, with two different values for *k* (*k=1 in the leftmost rule and k=2 for the rightmost rule*). The rule for *k=2* features 2 consequents since *k* measure can appear one or two times with probability greater than *t*. The rule in the middle of the figure captures the At least pattern, with *k=1*. Finally, the last rule captures the Exactly pattern, that checks that the k measure holds exactly 2 times (*k=2*).

Pause, End, Timeout and Simultaneously are four patterns included in the Time Category in the Quantitative Pattern catalog. Timed patterns are focused on particular instant in the system's execution and are canonical examples of probabilistic behavior since they occurrences are easier to predict than other events [21]. Figure 6 shows FVS rules addressing these timed patterns assuming a desired probability higher than *pr*. The End and Timeout patterns are shown in the top of Figure 6. When a timeout event occurs, the TimeOut pattern established that the execution must end. The end pattern is an extension of this behavior, since it models than the execution should end

after the occurrence of any event, and not only a timeout event. In the case of Figure 6, we introduced the *endEvent* to represent this behavior. In both cases, as soon they occur, the system will end with probability higher than *pr*. The FVS rule in the middle of the figure addresses the PAUSE pattern. In this case, the system resumes execution after pause with probability higher than *pr*. Finally, the rule at the bottom states that events e1, e2 and e3 must occur simultaneously with probability higher than *pr*.

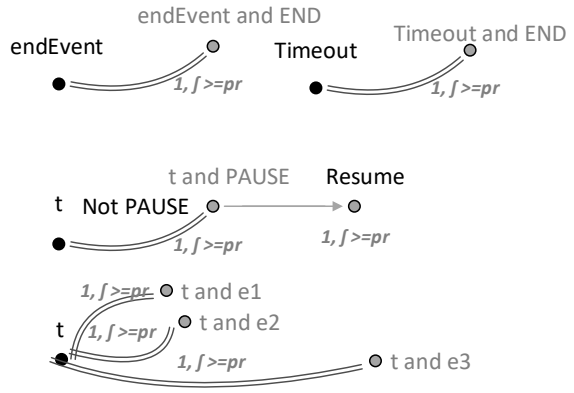

Figure 6. Some Timed Quantitative Patterns modeled in FVS

Preservation and Confidently patterns are concerned about maintaining a certain measure (preservation) or guarantying an upper bound of confidence in some specific values. These two patterns are shown next in Figure 7. The rule in the upper part targets the Preservation pattern (value *t* is preserved) while the other rule targets the Confidently pattern (measure *t* is ensured to be confident enough with probability greater than *pr*)

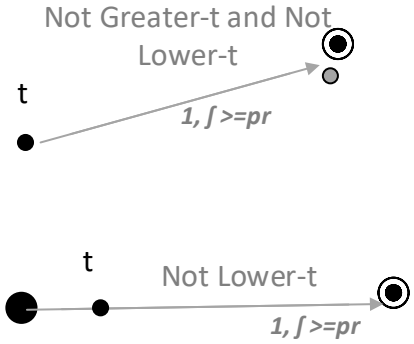

Figure 7. Preservation and Confidently Patterns in FVS

The Strictly Within and Within patterns specify that a measure value *t* will remain in a particular bound with a given probability. In the Within pattern the outer most value of the boundary is not mandatory to occur. These two patterns are shown in Figure 8 (The Within pattern above and the Strictly Within at the bottom)

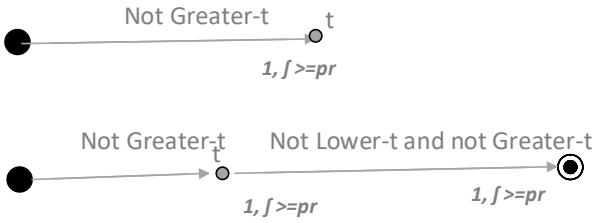

Figure 8. Boundary Patterns in FVS

3.2 Probabilistic version of Traditional Patterns in FVS

In this subsection we model in FVS the probabilistic patterns introduced in [22]. These patterns are a more direct version of the original patterns introduced by [24] adding probabilistic issues than the quantitative patterns described in Section 3.1.

Two interesting patterns are the Transient and the Steady pattern since they capitalize situations where some conditions must remain true or false given certain probabilities. The Transient pattern specifies that given the occurrence of an event *e* some property *p* must hold with a probability equal or greater than *pr*. The Steady pattern says that in the long run, a property *p* must hold with a probability equal or greater than *pr*. To model this pattern in FVS we simply state that the property *p* must occur in any future, where Any stand for the occurrence of any event in the system. This is the classical representation of a liveness property in a linear time behavioral specification. Both patterns are shown in Figure 9 (the Transient pattern in the top and the Steady pattern at the bottom).

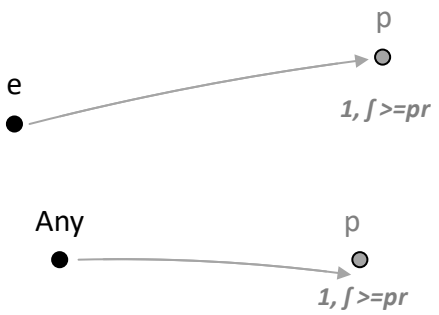

Figure 9. Transient and Steady Patterns in FVS

Following the probabilistic pattern work in [22] we model the next patterns as properties to be held between the occurrence of two events *t1* and *t2*. This corresponds to the notion of Between scope of a pattern [24], which state that a property must hold between two specific points instead of the entire execution. However, the patterns can be modeled with other scopes (for example, After or Before) or without any scope at all.

The Invariance pattern state that a property *p* must always be valid in a between points *t1* and *t2*. We consider that this must apply with probability equal or higher than *pr* but this can be easily adapted to shape other behavior. The existence pattern is very similar, but the property just need to occur for at least one instant instead of remaining true in the scope. These two patterns are shown in Figure 10 (Invariance in the top of the figure and Existence at the bottom of the Figure).

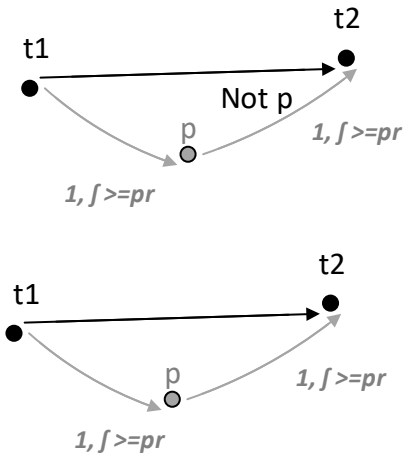

Figure 10. Invariance and Existence Patterns in FVS

Work in [22] also introduces a probabilistic version of the classic Response pattern, where an event response *r* must always follow a stimuli *s* with a given probability. The Constrained response is very similar, but some events must not occur until the response is found. These two patterns are shown in Figure 11. For the constrained version at the bottom of the figure we specify that event *z* must not occur until the occurrence of the response *r* with probability equal or higher than *pr*.

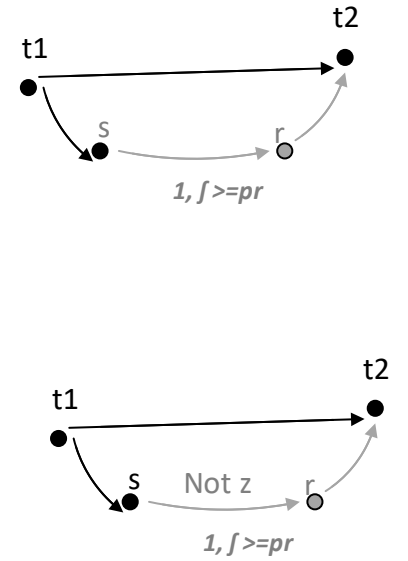

Figure 11. Response and Constrained Response Patterns in FVS

We conclude this section showing two last patterns: The Precedence pattern and the Until patterns. Although all the patterns introduced in [22] were modeled in FVS the portion of them shown in this section are representative enough. Similar to the previous ones, properties must hold between two instants, namely *t1* and *t2*. The Until pattern is shown at the top of Figure 12. The property *q* must hold until the occurrence of instant *t2* and after *t1*, with probability equal or higher than *pr*. The precedence pattern state that every response *r* must be preceded by a stimuli *s* with probability equal or higher than *pr*.

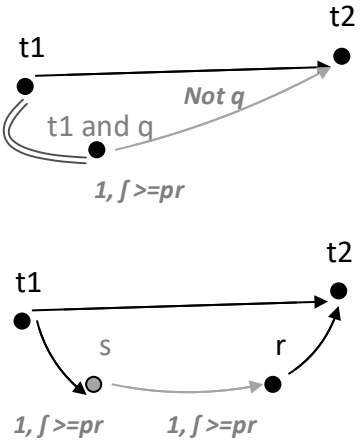

Figure 12. Until and Precedence Patterns in FVS

4. Case of Study: CubeSat Mission Specification

Among nano-satellites, the CubeSat project has emerged lately as one of the most popular ones [29]. The CubeSat program was initiated at Stanford University in nineteen ninety-nine for the purpose of building a low-cost and low-weight satellite. Over a thousand different CubeSats missions have been launched over the past twenty years [29, 33]. We developed a meaningful case of study consisting in a full document describing the expected behavior of a CubeSat satellite mission. It is conceived as a general specification template since additional requirements should be added to address a specific mission. It is a very complete document of more than thirty pages specifying the behavior of several satellites artifacts and components such as Orbital Parameters, Ground Storage Directions, Telemetry Functions, Propulsion Subsystem, Attitude Control Subsystem, Command Reception and Execution, Thermal Control Subsystem, Electrical Power Subsystem, Mechanical Subsystem, Dynamic Satellite Simulator or Data Handling Subsystem, among others. The requirements were obtained from real CubeSat missions

taken from the literature and represent real specification assignments [33-36].

In addition, we consider further requirements that suit perfectly for probabilistic measures since data can be easily collected from past telemetry missions. Some of them include: reaction wheel saturation and desaturation, solar panels energy optimization and satellite orbital correction maneuvers techniques.

We specified most of the requirements employing both kind of probabilistic patterns described in Section 3. The next subsection details how probabilistic values were added to our case of study.

4.1 Obtaining Probabilities for the Case Study

A key aspect when introducing probabilities to a behavioral specification model is how to obtain those probabilities. It is not the same stating that the server will response each request with a probability equal or higher than 0,99 than saying that the that the server will response each request with a probability equal or higher than 0,70. The probabilities have a direct impact on the decisions are made, how the execution flow and which events might or might not occur. This imply that a behavioral property might or might not be fulfilled according the probabilities assigned to each behavior of the system.

An attractive approach commonly used in probabilistic models is presented in [16]. This approach introduces probabilities based on how the user interacts with the system based on user profiles. In simpler words, the probabilistic are based on how the system has been used. However, the problem of obtaining probabilities might remain since producing and exploring this "user´s interaction profile" is not a trivial process [17,38,52].

For building "user-profile interaction" for our case of study we obtained values from existing and simulated nano satellites missions from several sources. The analysis of these data allowed us to build proper and accurate "user-profile interactions" and establish meaningful probabilities to each action. Two of our sources were missions with similar objectives than our case of study taken from The European Space Agency (ESA) website[1] and The European Cooperation for Space Standardization[2]. The other two sources were simulated missions obtained from different web sites and tools. The free trial version of the tool in https://www.ansys.com/products/missions/ansys-stk allows to obtain significant data that helped in a significant way to build our profiles and probabilities. We would like to strengthen these results obtaining a license of this tool and expanding far than the services provided in the trail version. The final source is an appealing simulating software that can be easily downloaded and installed from https://github.com/nasa/GTM_DesignSim . By combining the data recollected from these four sources we developed the "user-profile interactions" for our case of study.

[1] https://www.esa.int/
[2] https://ecss.nl/

5. Probabilistic FVS patterns application in the case of study.

In the next two sub sections we applied Probabilistic FVS behavioral pattern specification for our case of study.

5.1 Quantative Patterns in Action for Satellite Mission

We employed the Maximize Quantitative Pattern regarding solar energy optimization for satellite's solar panels. One of the requirements is 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.* To specify this requirement in FVS we rely on the EclipseDetected event and the BusVoltageMax to represent the measure to be maximized. The FVS rule instantiation of the Maximized Pattern is shown in Figure 13.

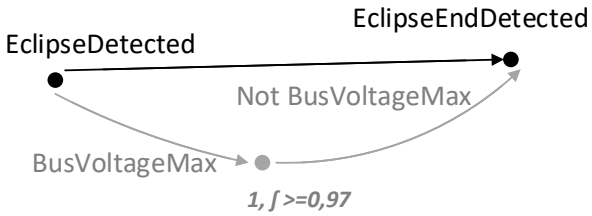

Figure 13. Maximizing Solar Panel energy

Another 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 as it is shown in Figure 14. The *CU* acronym used in the events in Figure 14 stands for Critical Units.

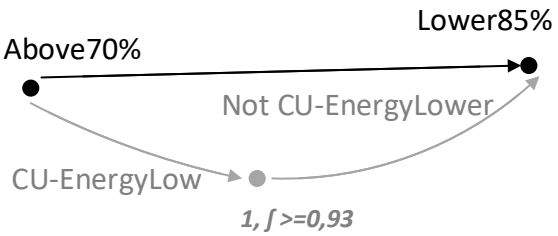

Figure 14. Minimizing CU Energy

The Preservation pattern can be employed for the following requirement: *The TTC subsystem shall continue to operate functionally down to a voltage 80 % of worst-case bus voltage with a probability equal or greater than 0,99*, where TTC stands for the Telemetry, Tracking and Command system. This is addressed in Figure 15.

Figure 15. Preservation Pattern for TTC operational voltage

To conclude the application of the quantitative patterns we illustrate the At Least pattern with the following requirement: *Pressure vessels shall perform with a burst pressure of at least 1.25 times the maximum expected operating pressure with a probability equal or greater than 0,99*. This is shown next in Figure 16.

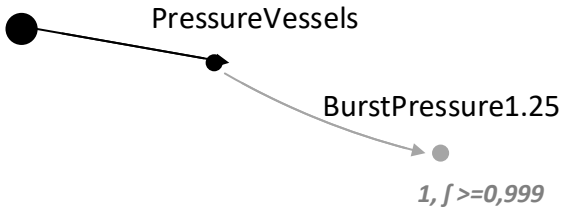

Figure 16. At Least Pattern for the pressure vessels measure

5.2 Traditional Patterns with Probabilities Application

In this section we illustrate how the traditional patterns with probabilities are instantiated in our case of study of Probabilistic Cube-Sat mission requirements.

The first example is the application of the Existence pattern to shape the correct desaturation of the reaction wheels' satellite's. When the reaction wheels reach its saturation limit desaturation must occur. Analyzing the telemetry data, we obtain that this behavior occurs with a probability equal or greater than 0.91. This is shown in the FVS rule in Figure 17.

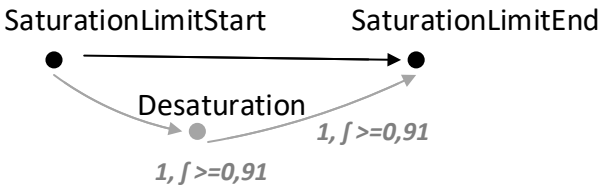

Figure 17. Reaction Wheels desaturation behavior

The Steady and the Transient patterns can be instantiated to shape requirements from the solar panel. For example, the solar panel temperature must be steady after orbiting is a requirement that is directly modeled by the Steady pattern with a probability equal or higher than 0.88. We employed the Transient pattern to model the following requirement: *The energy consumption must always be above 80% in the Starting mode* with a probability equal or higher than 0.99. This is shown in Figure 18.

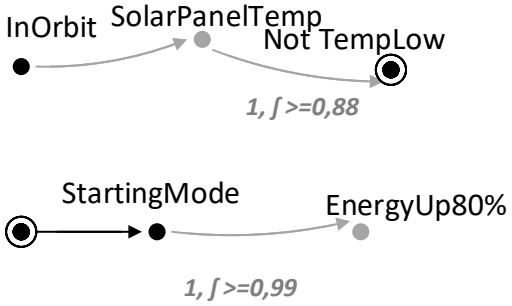

Figure 18. Transient and Steady Satellite requirements

We now model the following requirement: *The Attitude and Orbit Control Subsystem (AOCS) must monitor the control performance and attitude sensors until the final de-orbiting maneuver* with a probability greater than 0,99. This can be shaped with the Until Probabilistic pattern as it is depicted in Figure 19.

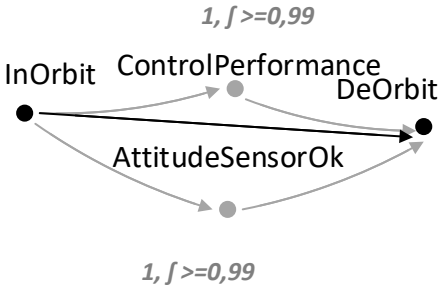

Figure 19. The Until Pattern in the Cube-Sat Example

Two more patterns application are shown next regarding the **Onboard Processor** behavior. First, we exemplified the Precedence pattern to model this requirement: *RAM contents are checked for errors before the processor is allowed to begin the reprogramming task execution* with a probability greater than 0,99. Lastly, we employed the Response pattern to state that between sensors activation every alarm must be followed by controller activation. These are shown in Figure 20.

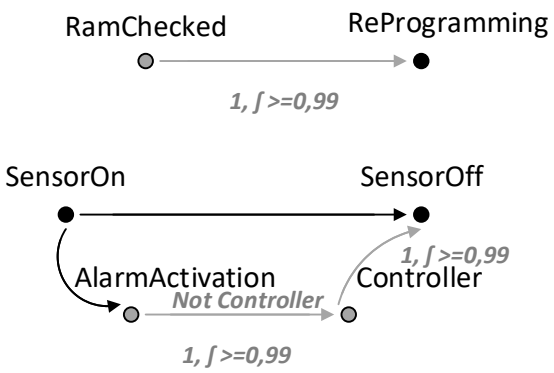

Figure 20. Requirements for the OnBoard Processor with Probabilities

6. Related Work

Work in [22] introduces the probabilistic version for the original specification patterns [24]. The repository of probabilistic patterns is called ProProST (Probabilistic Property Specification Templates). Besides the traditional patterns such as Response or Existence, the ProProST catalog include other patterns such as the Steady Pattern, the Transient pattern and the Invariant pattern. This set of three patterns constitute a nice feature since characteristics such as steadiness or transient are directly related to quantitative features [20].

Patterns are translated into a formal logic named CSL (Continuous Stochastic Logic) [39-40]. CTL is a classic option for continuous time models specification and it distinguish from other logics for introducing the *S* operator, capturing the notion of steadiness in long term runs of system's execution. In [22] it is claimed that the patterns could also be translated into other probabilistic temporal logics such as PCTL (Probabilistic Computation Tree Logic) [23] and PCTL* [41]. Finally, they also introduce a structured English grammar to formulate natural-language quality properties and requirements based on the developed specification patterns. The usage of this grammar eases the property specification process. In probabilistic FVS patterns are expressed through graphical scenarios that might result easier to explore and adapt to new scenarios [25].

PCTL (Probabilistic Computation Tree Logic) [23] and PCTL* [41] are two probabilistic logics that can be used to specify properties for discrete time models. Both logics are originated from the branching tree logics CTL and CTL* by exchanging the path quantifier ∃ and ∀ with a probabilistic operator *P*. Similarly, Probabilistic reward computation tree logic (PRCTL) [23]. PRCTL properties are interpreted over discrete-time Markov reward models, i.e., state machines containing states labelled with probabilities and rewards. FVS has been proved to be more expressive than regular temporal logics such as Linear Temporal Logic [32]. It would be interesting to explore the expressive power of probabilistic FVS with respect to these probabilistic logics.

Two widely known probabilistic model checkers are PRISM [42] and ETMCC [43]. One advantage of this numerical/symbolic models is their precision, but its main disadvantage is that they suffer from the state-explosion problem and high computational complexity [22].

Probabilities have been also intensively applied in cutting edge domains such as robotic and satellite missions. Work in [44] presents a very interesting set of patterns designed exclusively for shape the behavior of robotic missions. In [28] those patterns are also applied to satellite missions. These patterns for robotic missions are also extended in [21] to introduce probabilistic and quantitative aspects of formal requirements. As a distinguishable feature both quantative and robotic patterns can be combined constituting a powerful tool to specify behavior. Both kind of patterns can be also specified in probabilistic FVS with a distinctive graphical and declarative flavor. In addition, FVS behavioral specification can also be synthesized [26], which could add some potential to our approach.

Paper in [47] presents a novel technique for the formal analysis of distributed systems synthesizes formal specification and runtime verification, with specifications expressed in a domain-specific language rooted in first-order logic. They presented a challenging case of study using telemetry data from a satellite CAN bus (developed by INVAP S.E.), a mission-critical aerospace system. It would be more than interesting to explore if probabilistic FVS could be deployed in this context.

Self-Adaptive systems [8-10] can also be handled with probabilistic models due to the nature of dealing with uncertainty and adaptation. Some approaches in these directions are [17,45,46]. In [17] a fully functional framework denominated ENTRUST is presented. performs most of the verification tasks relying in known model checkers such as UPPALL [48] and PRISM [42] and a runtime probabilistic model checker to verify quantitative assertions [17]. Work in [45] shows the realization of adaptable multi-robot systems, which are capable of dealing with uncertainties by adapting their mission at runtime relying on a construction they denominate "adaptable task". Their model is validated employing two cases of studies based on real robot behavior. Finally, work in [46] present an approach for the automated reconfiguration at runtime in the robotics domain.

7. Future Work

We would like to continue this line of research including some of the following aspects. First of all, we would like to present formal proofs to establish that Probabilistic FVS is sound and complete. This is a natural next step and it simply implies extending the proofs of soundness and completeness of FVS [25] to cope with probabilities.

Secondly, we would like to incorporate the notion of rewards into our model [23]. Rewards is a key concept to enrich probabilistic models. We believe this could help to deepen the impact and contributions of Probabilistic FVS.

Thirdly, we would like to explore and compare FVS expressive power against other known approaches like named CSL, PCTL among others [39-41]. This exploration could help to understand which formalisms fits better under different circumstances and domains.

Fourthly, we are interested in synthesizing behavioral patterns. Behavioral synthesis [51] is an appealing technique that builds software that fulfills its specification by construction under the aegis of game theory algorithms [51]. FVS has been already explored in the behavioral synthesis world [25] so it would be to synthesize probabilistic patterns.

Finally, we would like being able to express conditional probabilities into our behavioral model. This option has been successfully explored in approaches like [49,50]. We believe this would further enrich FVS expressiveness to handle uncertainty through the use of probabilities since conditional statements are certainly powerful in this domain.

8. Conclusions

In this work it is exhibited the potential, flexibility and expressive power of the Probabilistic FVS language to specify behavior with probabilistic flavor. Probabilistic-FVS is capable enough to express in a declarative and graphical manner two important and different catalog of quantitative and probabilistic specification patterns. Both catalogs are shown in action in an appealing case of study, consisting of real requirements for a Cube-Sat nano-satellite mission. In addition, we build a sophisticated user-profile interaction to incorporate real and meaningful probabilities measures in our analysis.

References

[1] Bakota, T., Hegedűs, P., Körtvélyesi, P., Ferenc, R., & Gyimóthy, T. (2011, September). A probabilistic software quality model. In 2011 27th IEEE International Conference on Software Maintenance (ICSM) (pp. 243-252). IEEE.

[2] Hegedűs, P., Kádár, I., Ferenc, R., & Gyimóthy, T. (2018). Empirical evaluation of software maintainability based on a manually validated refactoring dataset. Information and Software Technology, 95, 313-327.

[3] Hérault, T., Lassaigne, R., Magniette, F., & Peyronnet, S. (2004, January). Approximate probabilistic model checking. In International Workshop on Verification, Model Checking, and Abstract Interpretation (pp. 73-84). Berlin, Heidelberg: Springer Berlin Heidelberg.

[4] Agarwal, C., Guha, S., Křetínský, J., & Pazhamalai, M. (2025). PAC statistical model checking of mean payoff in discrete-and continuous-time MDP. Formal Methods in System Design, 66(2), 195-237.

[5] Shooman, M. L. (1972). Probabilistic models for software reliability prediction. In Statistical computer performance evaluation (pp. 485-502). Academic Press.

[6] Godefroid, P., de Halleux, P., Nori, A. V., Rajamani, S. K., Schulte, W., Tillmann, N., & Levin, M. Y. (2008). Automating software testing using program analysis. IEEE software, 25(5), 30-37.

[7] Ali, H. M., Hamza, M. Y., & Rashid, T. A. (2024). A comprehensive study on automated testing with the software lifecycle. arXiv preprint arXiv:2405.01608.

[8] Weyns, D., Iftikhar, M. U., De La Iglesia, D. G., & Ahmad, T. (2012, June). A survey of formal methods in self-adaptive systems. In Proceedings of the fifth international c* conference on computer science and software engineering (pp. 67-79).

[9] Krupitzer, C., Roth, F. M., VanSyckel, S., Schiele, G., & Becker, C. (2015). A survey on engineering approaches for self-adaptive systems. Pervasive and Mobile Computing, 17, 184-206.

[10] Pekaric, I., Groner, R., Witte, T., Adigun, J. G., Raschke, A., Felderer, M., & Tichy, M. (2023). A systematic review on security and safety of self-adaptive systems. Journal of Systems and Software, 203, 111716.

[11] Carwehl, M., Vogel, T., Rodrigues, G. N., & Grunske, L. (2023). Runtime Verification of Self-Adaptive Systems with Changing Requirements. arXiv preprint arXiv:2303.16530.

[12] Vogel, T., Carwehl, M., Rodrigues, G. N., & Grunske, L. (2023). A property specification pattern catalog for real-time system verification with UPPAAL. Information and Software Technology, 154, 107100.

[13] Rodrigues, A., Caldas, R. D., Rodrigues, G. N., Vogel, T., & Pelliccione, P. (2018, May). A learning approach to enhance assurances for real-time self-adaptive systems. In Proceedings of the 13th International Conference on Software Engineering for Adaptive and Self-Managing Systems (pp. 206-216).

[14] Zudaire, S., Gorostiaga, F., Sánchez, C., Schneider, G., & Uchitel, S. (2021, May). Assumption monitoring using runtime verification for UAV temporal task plan executions. In 2021 IEEE International Conference on Robotics and Automation (ICRA) (pp. 6824-6830). IEEE.

[15] Bonnet, J., Gleizes, M. P., Kaddoum, E., Rainjonneau, S., & Flandin, G. (2015, September). Multi-satellite mission planning using a self-adaptive multi-agent system. In 2015 IEEE 9th International Conference on Self-adaptive and Self-organizing Systems (pp. 11-20). IEEE.

[16] Cailliau, A., & Lamsweerde, A. V. (2019). Runtime monitoring and resolution of probabilistic obstacles to system goals. ACM Transactions on Autonomous and Adaptive Systems (TAAS), 14(1), 1-40.

[17] Calinescu, R., Weyns, D., Gerasimou, S., Iftikhar, M. U., Habli, I., & Kelly, T. (2017). Engineering trustworthy self-adaptive software with dynamic assurance cases. IEEE Transactions on Software Engineering, 44(11), 1039-1069.

[18] Solano, G. F., Caldas, R. D., Rodrigues, G. N., Vogel, T., & Pelliccione, P. (2019, May). Taming uncertainty in the assurance process of self-adaptive systems: a goal-oriented approach. In 2019 IEEE/ACM 14th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS) (pp. 89-99). IEEE.

[19] Chung, L., Nixon, B. A., Yu, E., & Mylopoulos, J. (2012). Non-functional requirements in software engineering (Vol. 5). Springer Science & Business Media.

[20] Bass, L., Clements, P., & Kazman, R. (2012). Software Architecture in Practice: Software Architect Practice_c3. Addison-Wesley

[21] Menghi, C., Tsigkanos, C., Askarpour, M., Pelliccione, P., Vazquez, G., Calinescu, R., & García, S. (2022). Mission specification patterns for mobile robots: Providing support for quantitative properties. IEEE Transactions on Software Engineering, 49(4), 2741-2760.

[22] Grunske, L. (2008, May). Specification patterns for probabilistic quality properties. In Proceedings of the 30th international conference on Software engineering (pp. 31-40).

[23] Puterman, M. L. (1990). Markov decision processes. Handbooks in operations research and management science, 2, 331-434

[24] Dwyer, M. B., Avrunin, G. S., & Corbett, J. C. (1999, May). Patterns in property specifications for finite-state verification. In Proceedings of the 21st international conference on Software engineering (pp. 411-420).

[25] Asteasuain, F., & Braberman, V. (2017). Declaratively building behavior by means of scenario clauses. Requirements Engineering, 22(2), 239-274.

[26] Asteasuain, F., Calonge, F., Dubinsky, M., & Gamboa, P. (2021). Open and branching behavioral synthesis with scenario clauses. CLEI Electronic Journal, 24(3), 1-1.

[27] Asteasuain F., Legaspi A. Freire, G, Machuca M. A Declarative and Probabilistic Behavioral Specification Language to Handle Uncertainty in Modern Software Development. CONAIISI 2024

[28] Asteasuain F, Acuña J, Machuca M. Easing the Behavior Specification for Satellite Missions. CACIC 2024.

[29] Saeed, N., Elzanaty, A., Almorad, H., Dahrouj, H., Al-Naffouri, T. Y., & Alouini, M. S. (2020). CubeSat communications: Recent advances and future challenges. IEEE Communications Surveys & Tutorials, 22(3), 1839-1862.

[30] Clarke, E. M. (1997). Model checking. In Foundations of Software Technology and Theoretical Computer Science: 17th Conference Kharagpur, India, December 18–20, 1997 Proceedings 17 (pp. 54-56). Springer Berlin Heidelberg.

[31] Gamma, E., Helm, R., Johnson, R., Vlissides, J., & Patterns, D. (1995). Elements of reusable object-oriented software. Design Patterns, 1, 417.

[32] De Giacomo, G., & Vardi, M. Y. (2013, August). Linear Temporal Logic and Linear Dynamic Logic on Finite Traces. In Ijcai (Vol. 13, pp. 854-860).

[33] Kulu E. Nanosatellite and cubesat database. https://www.nanosats.eu/

[34] Doyle, M., Dunwoody, R., Finneran, G., Murphy, D., Reilly, J., Thompson, J., ... & Hanlon, L. (2021, June). Mission testing for improved reliability of CubeSats. In International Conference on Space Optics—ICSO 2020 (Vol. 11852, pp. 2699-2718). SPIE.

[35] Luppen, Z. A., Lee, D. Y., & Rozier, K. Y. (2021). A case study in formal specification and runtime verification of a CubeSat communications system. In AIAA Scitech 2021 forum (p. 0997).

[36] Walsh, S., Murphy, D., Doyle, M., Reilly, J., Thompson, J., Dunwoody, R., ... & McBreen, S. (2021). Development of the EIRSAT-1 CubeSat through functional verification of the engineering qualification model. Aerospace, 8(9), 254.

[37] Kwiatkowska, M., Norman, G., & Parker, D. (2018). Probabilistic model checking: Advances and applications. Formal System Verification: State-of the-Art and Future Trends, 73-121.

[38] Cunningham, K., Cox, D., Murri, D., & Riddick, S. (2011, August). A piloted evaluation of damage accommodating flight control using a remotely piloted vehicle. In AIAA Guidance, Navigation, and Control Conference (p. 6451).

[39] Aziz, A., Sanwal, K., Singhal, V., & Brayton, R. (1996, July). Verifying continuous time Markov chains. In International Conference on Computer Aided Verification (pp. 269-276). Berlin, Heidelberg: Springer Berlin Heidelberg.

[40] Baier, C., Katoen, J. P., & Hermanns, H. (1999, August). Approximative symbolic model checking of continuous-time Markov chains. In International Conference on Concurrency Theory (pp. 146-161). Berlin, Heidelberg: Springer Berlin Heidelberg

[41] Aziz, A., Singhal, V., Balarin, F., Brayton, R. K., & Sangiovanni-Vincentelli, A. L. (1995, July). It usually works: The temporal logic of stochastic systems. In International Conference on Computer Aided Verification (pp. 155-165). Berlin, Heidelberg: Springer Berlin Heidelberg.

[42] Kwiatkowska, M., Norman, G., & Parker, D. (2004). Probabilistic symbolic model checking with PRISM: A hybrid approach. International journal on software tools for technology transfer, 6(2), 128-142.

[43] Hermanns, H., Katoen, J. P., Meyer-Kayser, J., & Siegle, M. (2003). A tool for model-checking Markov chains. International Journal on Software Tools for Technology Transfer, 4(2), 153-172.

[44] Menghi, C., Tsigkanos, C., Pelliccione, P., Ghezzi, C., & Berger, T. (2019). Specification patterns for robotic missions. IEEE Transactions on Software Engineering, 47(10), 2208-2224.

[45] Filippone, G., Piñera García, J. A., Autili, M., & Pelliccione, P. (2024, April). Handling uncertainty in the specification of autonomous multi-robot systems through mission adaptation. In Proceedings of the 19th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (pp. 25-36).

[46] Peldszus, S., Brugali, D., Strüber, D., Pelliccione, P., & Berger, T. (2025). Software reconfiguration in robotics. Empirical Software Engineering, 30(3), 1-51.

[47] Lopez Pombo, C; Suez E. Vaccarezza, J. Verification of distributed systems through the analysis of communication data: preliminary ideas. ASSE- JAIIO-2025

[48] Larsen, K. G., Pettersson, P., & Yi, W. (1997). UPPAAL in a nutshell. International journal on software tools for technology transfer, 1, 134-152.

[49] Legay, A., Lukina, A., Traonouez, L. M., Yang, J., Smolka, S. A., & Grosu, R. (2019). Statistical model checking. In Computing and software science: state of the art and perspectives (pp. 478-504). Cham: Springer International Publishing.

[50] Ji, M., Wu, D., Li, Y., & Chen, Z. (2013). Counterexample Generation for Conditional Probability in Probabilistic Model Checking. J. Comput., 8(12), 3272-3279.

[51] Klein, U., Piterman, N., & Pnueli, A. (2012, January). Effective synthesis of asynchronous systems from GR (1) specifications. In International Workshop on Verification, Model Checking, and Abstract Interpretation (pp. 283-298). Berlin, Heidelberg: Springer Berlin Heidelberg.

[52] Pavese, E., Braberman, V., & Uchitel, S. (2010, May). My model checker died! how well did it do?. In Proceedings of the 2010 ICSE Workshop on Quantitative Stochastic Models in the Verification and Design of Software Systems (pp. 33-40).

[53] Pavese, E., Braberman, V., & Uchitel, S. (2009, August). Probabilistic environments in the quantitative analysis of (non-probabilistic) behaviour models. In Proceedings of the 7th joint meeting of the European software engineering conference and the ACM SIGSOFT symposium on The foundations of software engineering (pp. 335-344).