Formal Validation of Software for Nano Satellite Missions

» **Dr. Fernando Asteasuain**
CAETI – Universidad Abierta Interamericana – Facultad de Tecnología Informática, Ciudad de Buenos Aires, Argentina.
fernando.asteasuain@uai.edu.ar
Universidad Nacional de Avellaneda.

Abstract

Space research industry has become one of the most successful domains in the last years. In particular, the development of nano satellites has emerged as a stunning field since its low costs of production. The software in charge of the satellite functioning must be carefully verified to check that system fulfills the expected behavior. In this work we provide a full, complete and declarative framework to formally validate software for nano satellite missions, including behavioral synthesis which is a distinguishable contribution in this field. When validating the satellite behavior we include requirements from different sources: on board computer, IoT protocols, operating system and mission properties. Our framework is based on the declarative and graphical language FVS (Feather Weight Visual Scenarios).

KEYWORDS: NANO SATELLITES, FORMAL VERIFICATION, BEHAVIORAL SYNTHESIS

Validación Formal de Software para misiones de nano satélites

Resumen

La industria aeroespacial representa uno de las áreas de mayor crecimiento y éxito de los últimos años. En particular, el desarrollo de los denominados nanos satélites se ha destacado por sus bajos costos de producción. El software encargado de monitorear y ejecutar las tareas del satélite debe ser formalmente verificado para asegurar su correcto funcionamiento. En este trabajo proponemos un framework completo, íntegro y declarativo para verificar formalmente el comportamiento de nano satélites, incluyendo la síntesis de comportamiento, que es una contribución valiosa y poco explorada en este dominio. Para la experimentación tomamos requerimientos de diferentes lugares como la computadora de a bordo, los protocolos de IoT de comunicación, requerimientos del sistema operativo y también de la propia misión satelital. Nuestro framework se basa en el lenguaje de especificación gráfico y declarativo denominado FVS (Feather Weight Visual Scenarios).

Recibido: Marzo 2023 / Aceptado: Mayo 2023 / Publicado: Junio 2023

PALABRAS CLAVE: NANO SATÉLITES – VERIFICACIÓN FORMAL – SÍNTESIS DE COMPORTAMIENTO

Introduction

A new era for space-related systems has begun [1-8,11]. According to a report by Morgan Stanley, the expected revenue from the space industry will reach $22 billion by 2024 and $41 billion by 2029 [2, 9]. In particular, the design and development of small satellites has emerged as a golden star desired by everyone given their low-costs involved in their production and deployment.

One of the most popular nano satellites is perhaps the one called CubeSats [2]. The CubeSat program was initiated at Stanford University in 1999 for the purpose of building a low-cost and low-weight satellite. Over a thousand different CubeSats missions have been launched over the past 20 years [2,10] covering several domains such as communications, earth remote sensing, space tethering, biology, astronomy, space weather exploration and planetary science [2], among others.

As any other systems, software and hardware for nano satellites must be carefully verified. In this sense, formal verification techniques such as model checking can be naturally applied. Several aspects of nano satellites development must be formally verified: their on board computers, protocols and sensors communications, satellites objectives and missions, hardware, and its operating systems. On Board computers (OBC) consist on several software packages in charge of monitoring battery status, communication channels, connectivity, geo location services, telemetry information, and many other useful services. Two of the most widely operating systems used on OBC are FreeRTOS[1] and KubOs[2]. As described on its website, FreeRTOS, originally created as a real time operating system for micro controllers, is distributed freely under the MIT open source license, and includes a kernel and a growing set of IoT libraries suitable for use across all industry sectors. It has been used for satellites in many projects [2,310,11]. Similarly, KubOS also provides several interesting capabilities for controlling satellites, including several open sources libraries.

Formal verification is a key phase in every software development, but in domains which are harder to test and verify such as satellite systems it becomes even more crucial. Furthermore, behavioral synthesis [12] is a powerful technique that can be also applied to satellites formal verification. When applying synthesis, the system is built in a way such its behavior fulfills the specification by construction. This is achieved by employing game theory concepts: a game between the systems versus the environment. A controller (of the system) is built for the specified behavior if a winning strategy is found, an strategy that leads the system to a winning state no matter what move the environment choose. Usually, the controller takes the form of an automaton that decides which action to take considering the inputs gathered by the system's sensors.

In this work we applied a powerful and expressive behavioral and synthesis framework called Feather **Weight Scenarios (FVS)** [13,16-18] to formally verify and synthesize the following nano satellite's aspects: **two different OBC operating systems and a relevant IoT sensor**. In previous work we analyzed one operating system and a communication protocol [13,18]. We now expand those experiments showing promising results.

1 https://www.freertos.org /
2 https://docs.kubos.com/1.21.0/index.html

FVS is a graphical specification language based on events, and features an expressive and simple notation. In FVS behavior can be denoted by both linear and branching properties, and is more expressive than classical temporal logics such as Linear Temporal Logic (LTL) [16]. FVS graphical scenarios can be translated into Büchi automata, enabling the possibility of interacting with model checker and synthesis tools like GOAL [14] or LTSA [15]. **Besides providing a complete specification we also obtain a controller for each example, demonstrating the FVS capabilities to synthesize behavior**.

The rest of this paper is structured as follows. Section 2 briefly presents the FVS framework. Section 3 details the FreeRTOS formal specification and verification employing FVS as the specification formalism. A controller for the system is obtained, proving the consistency of the specification. Section 4 analyzes the specification of a piezoelectric sensor, which is a common a key IoT artifact in nano satellites development. Our validation examples end in Section 5, where the KubOS operating system is specified. Section 6 introduces related and future work whereas Section 7 presents some final observations of this work.

2. Feather Weight Visual Scenarios

In this section we will informally describe the standing features of FVS [16, 17]. The reader is referred [16, 17] 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 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 [16].

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.

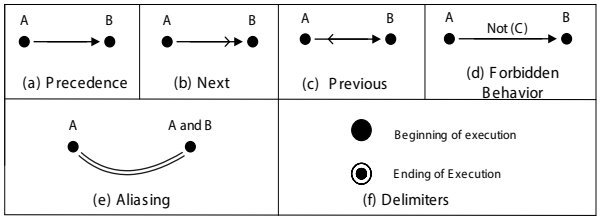

Figure 1. Basic Elements in FVS.

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 12-23
ISSN 2591-5320

2.1 Feather Weight Visual Scenarios

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. In other words, rules take the form of an implication: an antecedent scenario and one or more consequent scenarios.

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.

Two examples are shown in Figure 2 modeling the behavior of a client-server system. The rule in the top of Figure 2 establishes that every request received by a server must be answered, either accepting the request (consequent 1) or denying it (consequent 2). The rule at the bottom of Figure 2 dictates that every granted request must be logged due to auditing requirements.

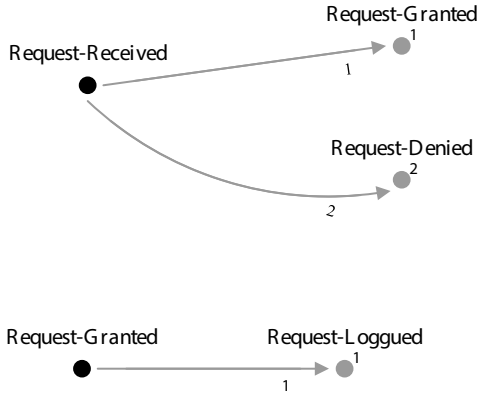

Figure 2. FVS rules.

2.2 Synthesizing Behavior in FVS

FVS specifications can be used to automatically obtain a controller employing a classical behavioral synthesis procedure. As explained in [12] this technique aims to find a winning strategy for our system. The opponent is the environment in which our system is deployed. Following the game metaphor, a winning strategy consists of a sequence of actions that our system can always take, no matter which "move" the environment make in every turn. This guarantees that the behavioral properties will always be satisfied. The winning strategy is displayed in an automaton shape called the controller. The controller is in charge of picking up the right action to take, following the winning strategy, in order to take the system to an accepting state. The formal specification is detailed in [12].

We now briefly explain how a controller is found in FVS while the complete description is available in [16,17].

Using the tableau algorithm detailed in [16,17] FVS scenarios are translated into Büchi automata. Then, if the obtained automata is deterministic, then we obtain a controller using a technique [19] based on the specification patterns [20] and the GR(1) subset of LTL. If the automaton is non deterministic, we can obtain a controller anyway. Employing an advanced tool for manipulating

diverse kinds of automata named GOAL [15] we translate these automata into Deterministic Rabin automata. Since synthesis algorithms are also incorporated into the GOAL tool using Rabin automata as input, a controller can be obtained.

Although this gain in expressiveness come with an cost in terms of performance due to the size of the involved automata we believe its crucial being able to express all type of behavioral properties.

3. FreeRTOS Specification and Controller in the FVS Framework

As explained in [1] FreeRTOS is a simple, easy-to-use real-time operating system. Its source code is written in C and assembly. It is open source and has little more than 2,200 lines of code. This operating system provides the following main services: task management, inter-task communication and synchronization, memory management, real-time events, and control of input and output devices.

Following the strategy adopted in [1] we specified FreeRTOS behavior taking into account its main functioning, the services it provides, and the invariants it must preserve. After modeling the expected behavior of the system we obtained a controller, thus proving the correctness of the FVS specification. Section 3.1 describes the FVS specification for the FreeRTOS system while Section 3.2 details how the controller for the system is automatically obtained.

3.1 FVS Specification for the FreeRTOS Operating System.

We first model FreeRTOS main requirements which describe its basic functioning. The systems must begin with a *boot* phase, which must be followed by a *scheduling* phase. This latter phase cannot begin unless the boot phase is finished. Once the scheduling phase is over, tasks can be executed. No tasks can begin its execution unless the scheduling phase is over. Four FVS rules are shown in Figure 3 to address these requirements. The first one says that the boot phase must be the first one to occur. We employed the event *anyOtherPhase* as a syntactic sugar simplification to avoid enumerating all the other possible phases. The second rule establishes that the scheduling phase must always follow the boot phase. The third rule in Figure 3 specifies that if the scheduling phase is active, then it must be the case that the system was previously in the boot phase. Finally, the last rule says that task can only be executed if and only of the *schedulingActive* event occurred previously.

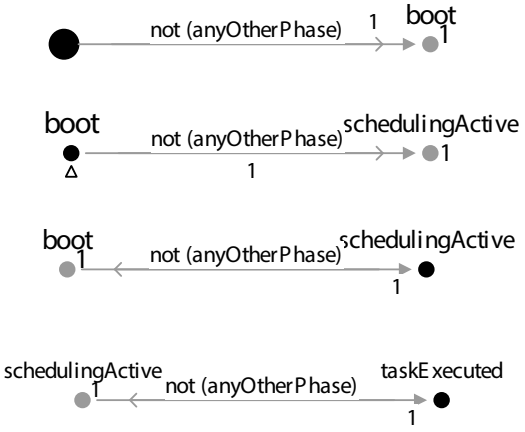

Figure 3. FVS rules describing freeRTOS main functioning

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 12-23
ISSN 2591-5320

Next we describe two rules shaping the behavior for FreeRTOS task management. The first requirement impose that a task can only be in one of four states: running, ready, suspended or blocked. It is addressed in the FVS rule in Figure 4.

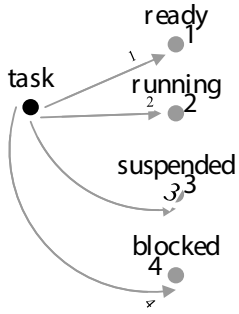

Figure 4. FVS rule for the task's possible states

The second requirement dictates how tasks are going to be picked by the scheduler. This is achieved in a classic priority scheme: the scheduler always chooses one task with the highest priority among the *ready* tasks. This is reflected in Figure 5.

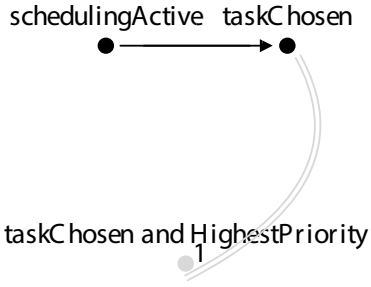

Figure 5. Scheduling Policy rule

We now define some rules for the communication and synchronization aspects of the FreeRTOS operating system. The first rule in this domain describes how information travels through the operating system tasks: employing queues. Tasks may post messages to queues and read messages from queues. This rule is shown in Figure 6.

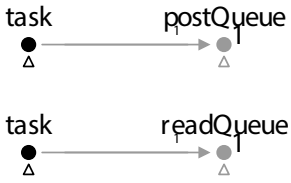

Figure 6. Tasks and Queues Behavior.

Finally, following the formalization of the system detailed in [1] we provide some rules describing the most two relevant invariants for the FreeRTOS operating system. The first invariant says that a task can only be in one state. The second one says that once the scheduler is active there is always a task running, either a regular task or the idle task. These two invariants are shown in Figure 7. As explained earlier, we employed a syntactic sugar simplification with the *anyOtherState* to avoid enumerating all the others possible state for each case.

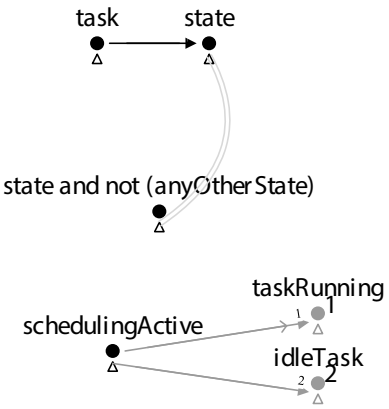

Figure 7. FreeRTOS invariants modeled in FVS

3.2 Obtaining a Controller

As explained in Section 2.2 a controller can be found using an FVS specification as input. Since a controller was found we can assert that there were no inconsistencies or unreachable states in the specification. The obtained automaton resulted in an automaton with 244 states and 467 transitions. A simplified version is shown of the controller in Figure 8.

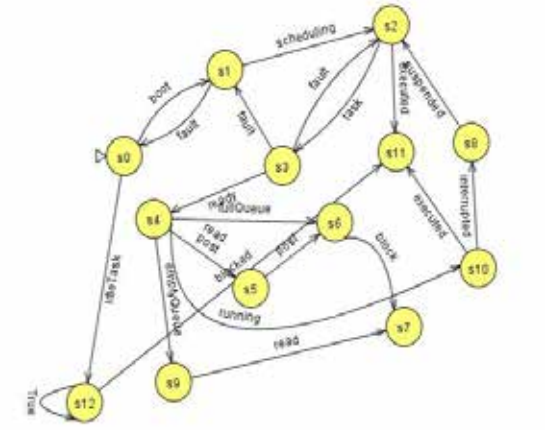

Figure 8. A simplified version of controller automaton for the FreeRTOS Operating System.

4. Piezoelectric Sensor Behavior: Specification and Controller

Typically nano satellite's information is gathered through the use of sensors. These devices capture the required data of the environment, which is afterwards transmitted to the satellite. Examples of data to be captured from the environment can be as varied as temperature, humidity, density, just to mention a few. One of the sensors employed in nano satellites is a piezoelectric sensor [22]. A piezoelectric is a device that uses the piezoelectric effect (the electric charge that accumulates in certain solid materials in response to applied mechanical stress) to measure changes in pressure, acceleration, temperature, strain, or force by converting them to an electrical charge [21].

When employing sensors formal validation and verification techniques must be applied in order to guarantee that data is not corrupted and that the sensor is accurately measuring. One technique is to have a mathematical model that can predict some expected output values from the sensor [22]. If the data received from the sensor is not predicted by the model, then an alarm must be raised indicating a faulty sensor.

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 12-23
ISSN 2591-5320

Following the behavior for a piezoelectric sensor indicated in [22] we obtained a FVS specification and its corresponding controller, including the usage of a mathematical model to detect anomalities. Some rules modeling the piezoelectric sensor are shown next. For example, Figure 9 models the expected outputs values from the sensor: *Stall*, *NoStall* and *Uncertain*.

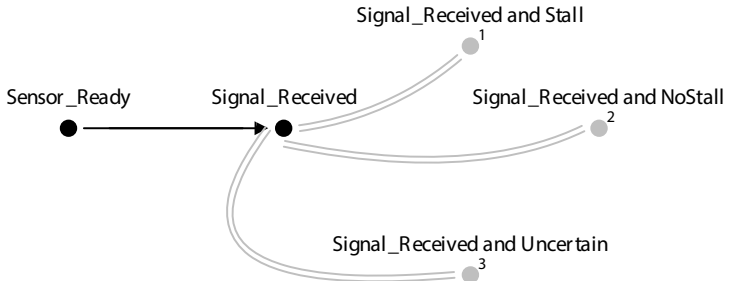

Figure 9. Expected outputs values from a piezoelectric sensor modeled in FVS

Figure 10 checks whether the information received from the sensor correspond to a possible value. This is achieved by designing a mathematical model which predicts the expected values in some range. If the received value belongs to the model, a *Signal_OK* event is triggered. Otherwise, an alarm is raised. The last rule in Figure 10 checks that the alarm is only raised if an out of range value is obtained. In this way, we verified that the alarm is raised only if an error had previosuly occurred.

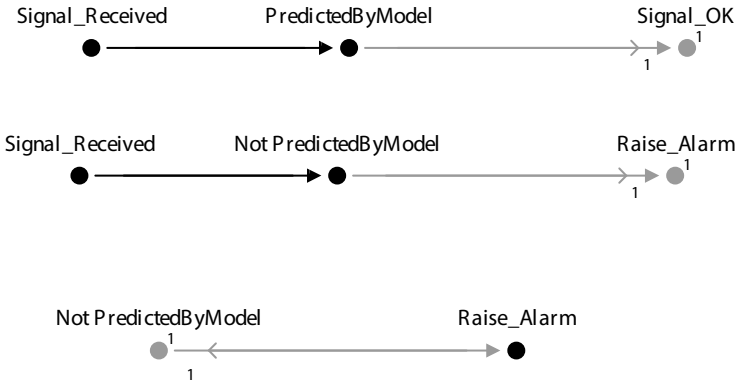

Figure 10. Checking the values obtained from the Sensor

Using the complete specification of the sensor we obtained a controller in form of a Rabin automaton. Figure 11 describes a portion of the controller for the piezoelectric sensor.

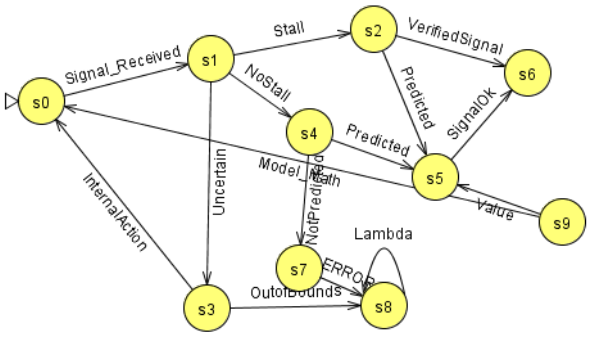

Figure 11. A controller for the piezoelectric sensor

5. KubOs Specification and Controller in the FVS Framework

We now specify in FVS some important aspects from the KubOS operating system. In particular, we focused on the Communication service, whose behavior description can be found in https://docs.kubos.com/1.15.0/tutorials/comms-service.html .

The basic behavior of this service can be described as: it is in charge of fetching incoming messages from the radio, parses its content forwards the message on and waits for a response. Once a response is received, it can be sent to the radio, to then be transmitted to the ground. If a response is not received, a timeout event is raised.

Two initial rules are shown in Figure 12. The rule at the top says that radio messages can only be received if the communication service is active. The rule at the bottom says that if the communication service is active and a radio message is received, then it must be the case that a proper service initialization occurred in between: the logging is initiated as well as the serial configuration, which is in charge of creating ports and connections. Note that the rule does not indicate a special order between both initializations, either Logging or serial can occur first.

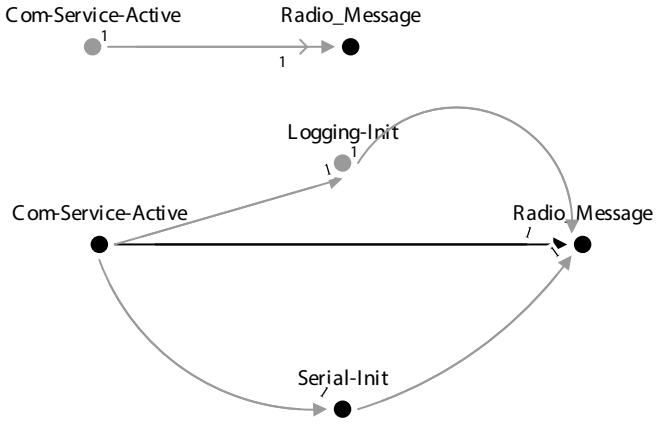

Figure 12. Communication Service Initial Rules

Three more rules are shown in Figure 13, establishing that messages must be parsed and then forwarded; that either a response or a timeout event can be received, and finally, that responses must only occur if a radio message was previously received.

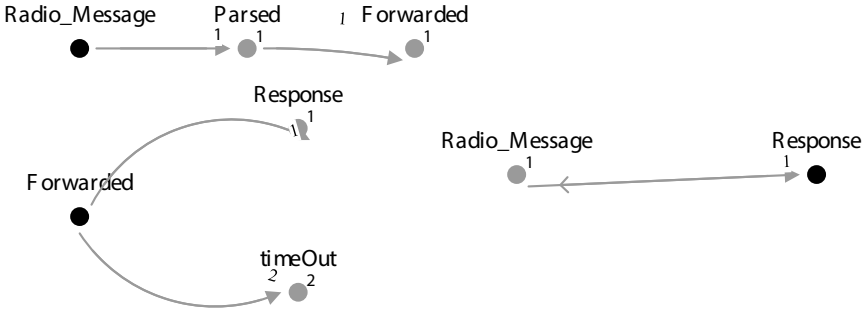

Figure 13. Communication Service Basic Behavior

Revista Abierta de Informática Aplicada/vol 7 Nº1 (2023): 12-23
ISSN 2591-5320

Finally, rules in Figure 14 show advanced behavior for the service: only when a response is received messages are sent to the radio and transmitted to the ground and ground communications can only occur if a response was previously received.

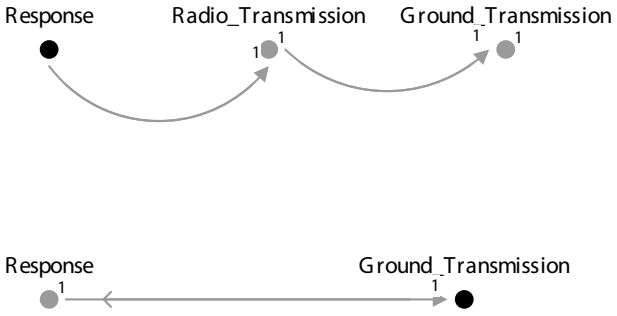

Figure 14. Communication Service Advanced Behavior

We conclude this section showing a part of the controller (Figure 15), which is automatically obtained upon the FVS specification.

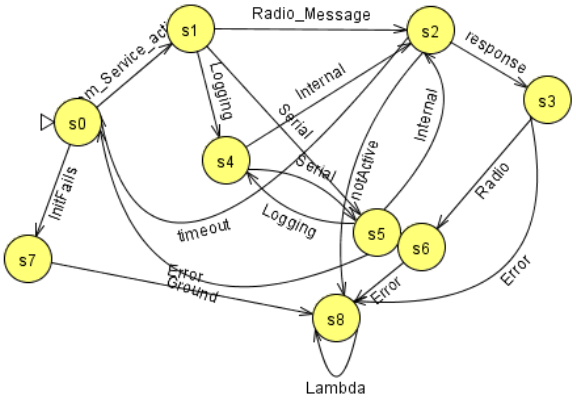

Figure 15. Sketch of a controller for the Communication Service as a Rabin Automaton

Related and Future Work

Perhaps work in [1] is the closest approach to the results presented in this paper. In [1] a formal specification for the FreeRTOS operating system is also given, describing the expected behavior of the system. The underneath formalism combines a complex state machine-based notation with a semi structured natural language notation, which resembles instructions in procedural languages. This formalism was difficult to learn, according to the results reported in [1]. On the other hand, FVS is a graphical and declarative notation, characteristics that ease the specification process. Also, the work in [1] is exclusively focused on the formal modeling of the behavior while our approach includes interaction with model checkers and behavioral synthesis, including obtaining a controller.

Work in [23] presents a very interesting implementation of the FreeRTOS operating system for embedded systems. We would certainly like to explore FVS's formal validation approach in this domain.

Other approaches analyzing formal verification over OBC satellites are [5-8]. A interesting future line of research is to compare FVS performance against all these approaches. It is worth mentioning that none of these approaches includes behavioral synthesis like our proposal.

In [22] a novel technique to specify safety properties in satellites and Data-Driven Flight Models is introduced. This work introduces the concept of "Safety envelopes", a special behavioral zone where some actions can be automatically performed in order to accommodate the satellite's status when a problem occurs. This is a very useful procedure since it allows to dynamically adapt the satellite's goals and objectives when the environment changes. Contrary to our work, this technique is based in probabilistic model checking. Other probabilities approaches are [23-25]. We would certainly like to explore FVS in this domain.

Conclusions

The design and development of nano satellites is a cutting-edge technology domain. These kind of systems need to be properly verified. This is a very challenging objective since they are particularly hard to test since it is very difficult to replicate satellites' environment conditions.

In this work we strengthen the FVS framework as a powerful tool to specify, verify and synthesize behavior for the development of nano satellites. We employed FVS to model two of the most common operating systems for on onboard computers. In addition, we specify the behavior of a widely used IoT sensor, namely a piezoelectric sensor. As a distinguishable feature besides specifying the expected behavior of the system we automatically obtain a controller for each example.

Referencias

» [1] Déharbe, D., Galvao, S., & Moreira, A. M. (2009, August). Formalizing freertos: First steps. In Brazilian Symposium on Formal Methods (pp. 101-117). Springer, Berlin, Heidelberg.

» [2] 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.

» [3] Hanafi, A., Derouich, A., Karim, M., & Lemmassi, A. (2021, January). Design and Implementation of an Open Source and Low-Cost Nanosatellite Platform. In International Conference on Digital Technologies and Applications (pp. 421-432). Springer, Cham.

» [4] Williams, C. and DelPozzo S. Nano Microsatellite Market Forecast - 10th Edition, Space-Works Annual Nano/Microsatellite Market Assessment, 2020.

» [5] Krishnan, R., & Lalithambika, V. R. (2020). Modeling and Validating Launch Vehicle Onboard Software Using the SPIN Model Checker. Journal of Aerospace Information Systems, 17(12), 695-699.

» [6] Aurandt, A., Jones, P. H., & Rozier, K. Y. (2022). Runtime verification triggers real-time, autonomous fault recovery on the CySat-I. In NASA Formal Methods Symposium (pp. 816-825). Springer, Cham.

» [7] Tipaldi, M., Legendre, C., Koopmann, O., Ferraguto, M., Wenker, R., & D'Angelo, G. (2018). Development strategies for the satellite flight software on-board Meteosat Third Generation. Acta Astronautica, 145, 482-491.

» [8] Wenker, R., Legendre, C., Ferraguto, M., Tipaldi, M., Wortmann, A., Moellmann, C., & Rosskamp, D.

(2017, June). On-board software architecture in MTG satellite. In 2017 IEEE International Workshop on Metrology for AeroSpace (MetroAeroSpace) (pp. 318-323). IEEE.

» [9] Pressman, A. (2019). Why Facebook, SpaceX and dozens of others are battling over Internet access from space. Fortune.

» [10] Kulu E.. "Nanosatellite and cubesat database," 2019. [Online]. Available: https://www.nanosats.eu/

» [11] Alam, M., Khamees, A., Aboelnaga, T., Amer, A., Harbi, A., Alamir, M., ... & Elsayed, O. A. (2021, August). Design and Implementation of an Onboard Computer and payload for Nano Satellite (CubeSat). In The International Undergraduate Research Conference (Vol. 5, No. 5, pp. 361-364). The Military Technical College.

» [12] D'Ippolito, N. R., Braberman, V., Piterman, N., & Uchitel, S. (2010, November). Synthesis of live behaviour models. In Proceedings of the eighteenth ACM SIGSOFT international symposium on Foundations of software engineering (pp. 77-86).

» [13] Asteasuain F. (2021). "Formalizing Operating Systems for nano satellites On Board Computers". 10mo. Congreso Nacional de Ingeniería Informática y Sistemas de Información CoNaIISI 2022.

» [14] Y.-K. Tsay, Y.-F. Chen, M.-H. Tsai, K.-N.Wu, and W.-C. Chan. Goal: A graphical tool for manipulating büchi automata and temporal formulae. In TACAS, pages 466-471. Springer, 2007

» [15] Uchitel, S., Chatley, R., Kramer, J., & Magee, J. (2003, April). LTSA-MSC: Tool support for behaviour model elaboration using implied scenarios. In International Conference on Tools and Algorithms for the Construction and Analysis of Systems (pp. 597-601). Springer, Berlin, Heidelberg.

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

» [17] Asteasuain, F, Calonge F, Dubinsky M, . Gamboa P. Open and branching behavioral synthesis with scenario clauses. CLEI ELECTRONIC JOURNAL, 24(3), 2021.

» [18] Asteasuain F. (2021). "Controller Synthesis for IoT Protocols Verification". 9no. Congreso Nacional de Ingeniería Informática y Sistemas de Información CoNaIISI 2021. Modalidad Virtual. Facultad Regional Mendoza. Universidad Tecnológica Nacional. Mendoza, 4 y 5 de Noviembre de 2021.

» [19] Maoz S. and Ringert J. O.. Synthesizing a lego forklift controller in gr (1): A case study. arXiv preprint arXiv:1602.01172, 2016.

» [20] Dwyer, M. Avrunin, M, and Corbett M.. Patterns in property specifications for finite-state verification. In ICSE, pages 411-420, 1999.

» [21] Tressler, J. F., Alkoy, S., & Newnham, R. E. (1998). Piezoelectric sensors and sensor materials. Journal of electroceramics, 2, 257-272.

» [22] Cruz-Camacho, E., Amer, A., Kopsaftopoulos, F., & Varela, C. A. (2023). Formal Safety Envelopes for Provably Accurate State Classification by Data-Driven Flight Models. Journal of Aerospace Information Systems, 20(1), 3-16.

» [23] Jackson, J., Laurenti, L., Frew, E., & Lahijanian, M. (2020, December). Safety verification of unknown dynamical systems via gaussian process regression. In 2020 59th IEEE Conference on Decision and Control (CDC) (pp. 860-866). IEEE.

» [24] Jeannin, J. B., Ghorbal, K., Kouskoulas, Y., Gardner, R., Schmidt, A., Zawadzki, E., & Platzer, A. (2015). A formally verified hybrid system for the next-generation airborne collision avoidance system. In Tools and Algorithms for the Construction and Analysis of Systems: 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015, Proceedings 21 (pp. 21-36). Springer Berlin Heidelberg.

» [25] Paul, S., Kopsaftopoulos, F., Patterson, S., & Varela, C. A. (2020). Towards Formal Correctness Envelopes for Dynamic Data-Driven Aerospace Systems. Handbook of Dynamic Data-Driven Application Systems, edited by Darema F. and Blasch E., Springer, Berlin.