# 10 mo CONGRESO NACIONAL

# CoNaIISI

Congreso Nacional de Ingeniería Informática / Sistemas de Información

## 2022 Modalidad Híbrida

Facultad Regional Concepción del Uruguay

UNIVERSIDAD TECNOLÓGICA NACIONAL

AJEA
Actas de Jornadas y Eventos Académicos de UTN

Universidad Tecnológica Nacional, Facultad Regional Concepción del Uruguay
10mo. Congreso Nacional de Ingeniería Informática y Sistemas de Información / compilación de Adrián Callejas ... [et al.]. - 1a ed. - Ciudad Autónoma de Buenos Aires : Universidad Tecnológica Nacional, 2022.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-950-42-0218-9

1. Tecnología Informática. I. Callejas, Adrián, comp. II. Título.
CDD 004.071


# 10 mo. Congreso Nacional de Ingeniería Informática / Sistemas de Información

**3 y 4 de Noviembre 2022**

Facultad Regional Concepción del Uruguay

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

# Formalizing Operating Systems for nano satellites On Board Computers

*Fernando Asteasuain* [1,2]
[1]*Universidad Nacional de Avellaneda- fasteasuain@undav.edu.ar*
[2]*Universidad Abierta Interamericana - Centro de Altos Estudios CAETI*

## Abstract

*In this work we provide a formal verification of the FreeRTOS operating system. Even most, a controller for the system is found. FreeRTOS is one of the most used operating systems for On Board Computers (OBC) in nano satellites such as CubeSats. Exploring, verifying and understating the potential and limitations of OBC operating systems is crucial for the growth of one of the most promising domains in the modern world: the space research industry. FreeRTOS's formal verification is achieved employing the Feather Weight Visual Scenarios (FVS) framework, which has been previously applied to verify Internet of Things (IoT) protocols.*

.

*Keywords: Formal Specification, CubeSats, OBC.*

## 1. 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].

These high expectations in this domain originated a competitive race in the space research field, both in the academic and in the industrial world. In this attractive and bubbly scenario 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. In the past ten years the nanosatellite segment grow by a factor of 10x, from as few as 20 satellites in 2011 to nearly 200 in 2019, and it is estimated that 1800 to 2400 nano/microsatellites will need to be launched over the next 5 years [3,4].

In particular, CubeSats, have turned out lately as one of the most popular microsatellites [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].

There are a plethora of different domains where CubeSats satellites are directly involved: communications, earth remote sensing, space tethering, biology, astronomy, space weather exploration and planetary science [2], just to mention a few.

Because of its designs goals CubeSats are mainly addressed for Low Earth Orbit missions [1-8,11]. This present a huge challenge regarding the interaction between the satellites and the Internet of Things (IoT) artifacts receiving and handling data and information, taking into account communications range and data rate. The challenges include a global IoT coverage, ensuring seamless communication with the IoT devices placed in rural and even remote areas. As fully detailed in [24], space-to-Earth communications are more challenging than Earth-to-Earth ones, mainly because the channel losses and the *Doppler* frequency shifts are higher and - possibly- because of the ionospheric effects.

Additionally, CubeSats' On BOARD Computers (OBC) must be carefully designed, planned, developed and verified. For example, battery consumption and other resources are extremely important since the services the satellites provide heavily rely on them. An OBC is in charge of the all the satellites services, it monitors them to ensure their health, and also monitors the status of all the subsystems. An OBC also interacts with the ground station to send the required telemetry data and satellite status [24,25].

The design of OBC has been declared of national relevance in Argentina and it has been included in the topics of the Plan Nacional de Ciencia, Tecnología e Innovación Argentina 2030.

One of the most important aspects to decide when developing a OBC is the operating system to be installed on it. Two of the most widely operating systems used 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.

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

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

Formal verification is a key phase in every software development, but in those domains which are harder to test and verify such as satellite systems it becomes even more crucial. Furthermore, behavioral synthesis [12,13] 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 focused on the formal verification of the FreeRTOS operating system for OBC in CubeSats satellites in a way to provide a better understanding of its possibilities, potential and limitation as a potential operating system for the space industry development. We decided to analyze this operating system because of its known use and also because its formalization has been proposed as a case of study in the International Grand Challenge on Verified Software [14,15].

As the specification formalism we employed a powerful and expressive behavioral and synthesis framework called Feather **Weight Scenarios (FVS)** [16-18]. 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 [19] or LTSA [20]. In [18] we explored the FVS formalism to verify one of the most relevant protocols for Iot: the MQQT[3] protocol. As IoT plays a key role for satellite communication, this previous work constitutes a solid milestone.

The rest of this work is structured as follows. Section2 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. Sections 4 and 5 present related and future work while Section 6 concludes the paper by presenting some final observations.

## 2. FVS: Feather Weight Visual Scenarios.

In this section we will informally describe the standing features of Branching FVS, a simple branching extension of the FVS language [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.

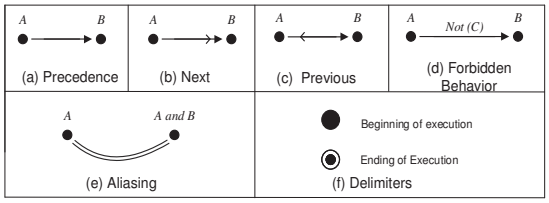


**Figure 1. Basic Elements in FVS.**

### 2.1 FVS rules

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.

[3] http://docs.oasis-open.org/mqtt/mqtt/v5.0/

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

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.

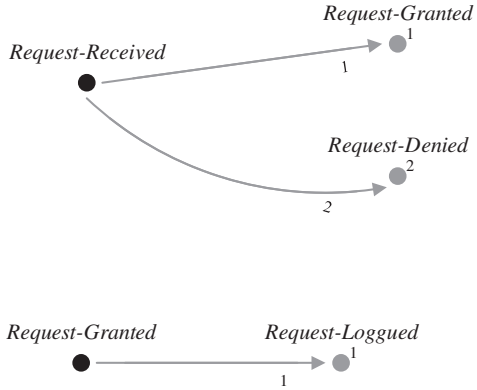


**Figure 2. FVS rules.**

### 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 [12,13] 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 [12,13].

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 [21] based on the specification patterns [22] 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 [19] 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.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

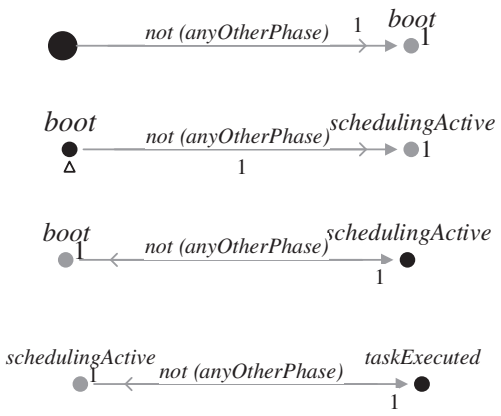


**Figure 3. FVS rules describing freeRTOS main functioning**

Next we describe 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.

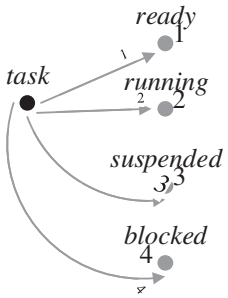


**Figure 4. FVS rule for the task's possible states**

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.

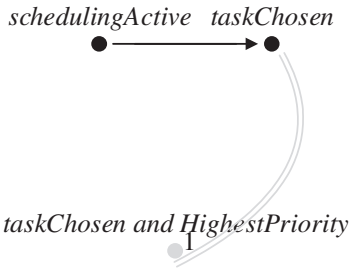


**Figure 5. Scheduling Policy rule**

The third requirement for task management involves the creation of an *idle* task. This task has the lowest possible priority. This task guarantees that the processor is always executing some task. The rule in Figure 6 deals with the idle task as required.

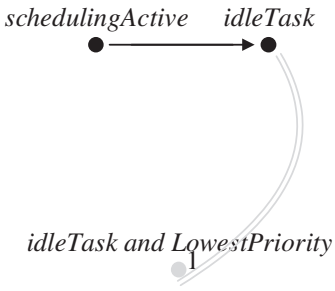


**Figure 6. The idle task modeled in FVS.**

The final requirement included in this specification regarding task management says that if there are two or more tasks having the highest priority among the ready tasks, then they shall equally share the processing time. This is tackled in the rule shown in Figure 7. It says that if two rules are executed sharing processing time then it must be the case they were assigned with the same priority.

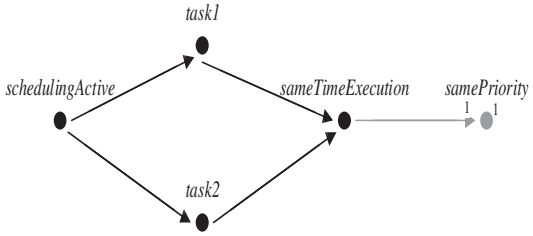


**Figure 7. Same priority policy in FVS.**

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 8.

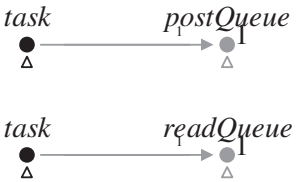


**Figure 8. Tasks and Queues Behavior.**

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

The following two rules correspond to a classic queue with limited capacity behavior: whenever a task wants to read from an empty queue or wants to write to a full queue, it is blocked. An attempt to post in a full queue is shown in Figure 9 while an attempt to read from an empty queue is shown in Figure 10.

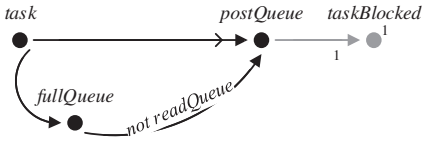


**Figure 9. Posting in a full queue**

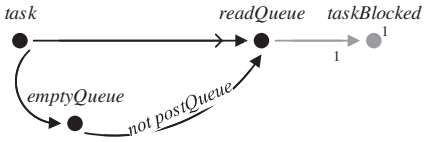


**Figure 10. Reading from an empty queue**

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 11. As explained earlier, we employed a syntactic sugar simplification with the *anyOtherState* to avoid enumerating all the others possible state for each case.

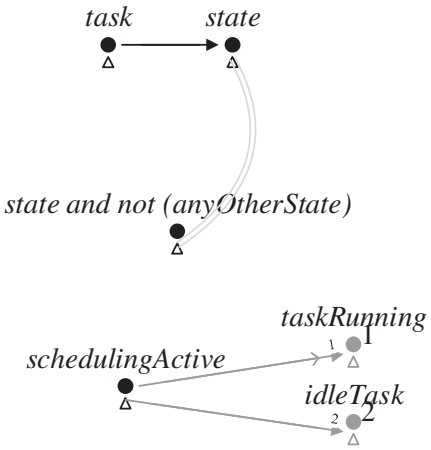


**Figure 11. FreeRTOS invariants modeled in FVS**

### 3.1 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 12.

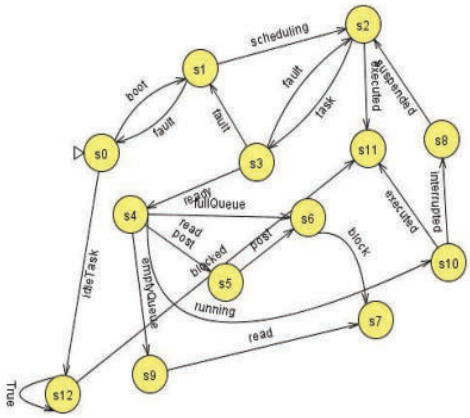


**Figure 12. A simplified version of controller automaton for the FreeRTOS Operating System.**

## 4. Related 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 RTOS 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.

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

## 5. Future work

Several lines of research may continue this work. In first place, we would like to combine FVS's controllers with automatic code generators, in order to obtain not also a model but also a possible implementation for the system. This would constitute a major advance in our research.

We would also like to compare FVS performance against other approaches [5-8]. This implies an empiric evaluation over a consolidated case of study.

Finally, we are interested in extending formal validation and verification to all the aspects involved in the services offered by CubeSats, and not only the OBC's operating system. These services include features such as communication, protocols and resources' consumption analysis.

## 6. Conclusions

The design of On Board Computers (OBC) for nano satellites like CubeSats is a key activity for the space research industry. The choice of the operating system of the OBC determines how much information the satellite is able to process and handle. Taking this into account we present a formal verification for the FreeRTOS operating system. FVS was able to specify all the expected behavior of the system, and a controller was found. We believe the graphical and declarative flavor of FVS may ease the exploration and analysis of the different aspects to be decided when designing a OBC.

## References

[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] Bloem, R., Jobstmann, B., Piterman, N., Pnueli, A., & Sa'ar, Y. (2012). Synthesis of reactive (1) designs. Journal of Computer and System Sciences, 78(3), 911-938.

[14] Jones, C., O'Hearn, P., & Woodcock, J. (2006). Verified software: A grand challenge. Computer, 39(4), 93-95.

[15] Woodcock, J., & Banach, R. (2007). The verification grand challenge. J. Univers. Comput. Sci., 13(5), 661-668.

[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 202

[19] 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

[20] 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.

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

AJEA - Actas de Jornadas y Eventos Académicos de UTN
DOI: https://doi.org/10.33414/ajea.1146.2022

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

[23] Lamichhane, K., Kiran, M., Kannan, T., Sahay, D., Ranjith, H. G., & Sandya, S. (2015, June). Embedded RTOS implementation for Twin Nano-satellite STUDSAT-2. In 2015 IEEE Metrology for Aerospace (MetroAeroSpace) (pp. 541-546). IEEE.

[24] Fernandez, L., Ruiz-De-Azua, J. A., Calveras, A., & Camps, A. (2020). Assessing LoRa for satellite-to-earth communications considering the impact of ionospheric scintillation. IEEE access, 8, 165570-165582.

[25] Woolley, M. (2019, January). Bluetooth core specification v5. 1. In Bluetooth.