9° Congreso Nacional

CoNaIISI 2021

de Ingeniería Informática y Sistemas de Información

4 5 NOVIEMBRE

confedi
Consejo Federal de Decanos de Ingeniería

UTN FACULTAD REGIONAL MENDOZA
UNIVERSIDAD TECNOLÓGICA NACIONAL

Departamento de Ingeniería en Sistemas de Información

Red de Ingenierías en Informática / Sistemas de Información
RIIISIC
ConFeDI

9° Congreso Nacional de Ingeniería Informática/Sistemas de Información

CoNaIISI

4 y 5 de noviembre de 2021

Universidad Tecnológica Nacional

Facultad Regional Mendoza

Memoria de Trabajos

9° Congreso Nacional de Ingeniería Informática/Sistemas de Información
9° CoNaIISI: Memorias de trabajos / Compilación de Marcela Fernandez; Matilde César; María Gabriela Martínez. - 1a ed. –
Ciudad Autónoma de Buenos Aires: Universidad Tecnológica Nacional, 2022.
Libro digital, PDF

Archivo Digital: descarga y online
ISBN 978-950-42-0213-4

1. Sistemas de Información. 2. Ingeniería Informática. I. Fernandez, Marcela, comp. II. César, Matilde, comp. III. Martínez, María Gabriela, comp. IV. Título
CDD 004.0711

Compilación de: Fernandez, Marcela
César, Matilde
Martínez, María Gabriela

Revisado por: Carbonari, Daniela
Caymes Scutari, Paola
Bianchini, Germán

ISBN 978-950-42-0213-4

Auspiciantes:

Controller Synthesis for IoT Protocols Verification

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

Abstract

Almost every modern device or artifact can now send and/or receive information and data. This hyper connected modern era is commonly denominated Internet of Things (IoT). Software Engineering tools and techniques must be adapted to manage the new challenges and requirements that the emergent paradigm of IoT imposes, especially regarding to communications, interactions and protocols between those artifacts. In this work we focus on a very well known formal technique called Controller Synthesis, which features interesting characteristics to formally verify Iot systems. In particular, we specified and synthesized the behavior of a key protocol for IoT, the MQQT protocol, employing the FVS formal verification framework.

Keywords: IoT, Formal Verification, Synthesis.

1. Introduction

The amazing growth of the so called Internet of Things (Iot) initiated a challenging revolution in the Software Engineering Community. Almost every device can now be integrated and controlled by a software system by adding sensors to it that capture and communicate information from the device to a centralized system. These new kind of integrated and reactive systems represent the leading role of emerging and cutting edge technologies such as Industry 4.0 [1] or Machine to Machine (M2M) interactions [2].

In these systems communications and interactions between artifacts play a crucial role. Therefore, analyzing, reasoning and verifying the protocols used to communicate artifacts are key activities [3-8]. Two of the most used protocols for the IoT systems are the Message Queue Telemetry Transport (MQTT[1]) and the Constrained Application Protocol (CoAP)[2].

MQQT protocol is based on the classical architectural pattern Publish/Subscribe whereas CoAp is based on a client/server fashion and is specially used in domains like such as smart energy and building automation [3, 4]. In the publish/subscribe architecture processes that produces the information or data publish in an intermediate buffer which in turn, passes the information to other processes interested in it. Processes interested in receiving the information must previously subscribe to the intermediate buffer.

A plethora of approaches aim to formally prove the correctness and soundness of IoT protocols [3-8, 21-23, 26]. In few words, these approaches apply tools such as modern model checkers to verify the expected behavior of the protocols. Several formalisms like probabilistic automata [24], temporal logics, Temporal Logics of Actions [16] or process algebra are employed [25].

However, to our best knowledge, a very well known formal verification technique called Controller Synthesis [9-10] has not been widely applied in this domain.

Controller synthesis offers attractive features to be considered in the IoT domain. First of all, it is heavily oriented to event-based systems. Secondly, it takes into consideration not only the system but also the environment where the system lives. This is called Open Systems [11], and actions are divided into controllable (those governed by the system) and uncontrollable (those governed by the environment).

When applying synthesizing algorithms 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. If a controller is found, there it is guaranteed that it fulfills the specification. Otherwise, conditions and behavior should be revisited in order to try a new version. 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 domains such as IoT Protocols where multiple factors arise such as hardware failures and/or, environment's unstable conditions, the possibility to build automatically the system with the strong

[1] http://docs.oasis-open.org/mqtt/mqtt/v5.0/
2 https://coap.technology/

guarantee that it fulfills the specification it is worth to be explored.

Given this context in this work we explore a powerful and expressive behavior and synthesis framework called Feather Weight Visual Scenarios (FVS) [12] in the IoT domain.

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) [12]. FVS graphical scenarios can be translated into Büchi automata, enabling the possibility of interacting with model checker and synthesis tools like GOAL [16] or LTSA [19].

In particular, we specified the behavior of the MQQT protocol and we automatically obtain a controller using known external tools.

The rest of this work is structured as follows. Section2 briefly presents the FVS framework. Section 3 shows FVS in action by specifying the behavior of the MTTQ protocol and detailing how a controller for the system is built. Section 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 [12]. The reader is referred [12] 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 [12].

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.

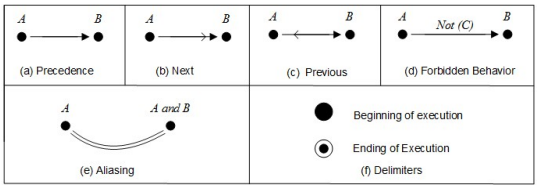

Figure 1. Basic Elements in FVS.

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.

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.

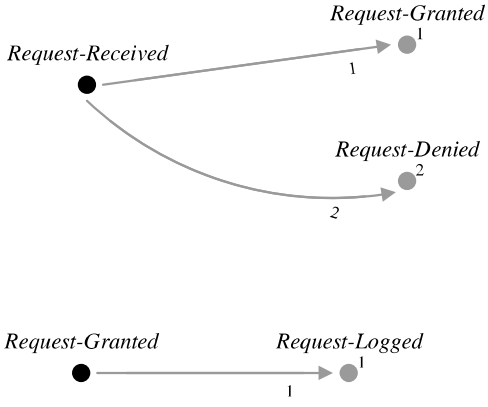

Figure 2. FVS rules.

2.2 Synthesizing Behavior in FVS

FVS specifications can be used to automatically obtain a controller employing a classical behavioral synthesis procedure. We now briefly explain how this is achieved while the complete description is available in [27-28].

Using the tableau algorithm detailed in [12] FVS scenarios are translated into Büchi automata. Then, if the obtained automata is deterministic, then we obtain a controller using a technique [13] based on the specification patterns [14] 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 [16] 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. A controller for the MQTT protocol

In this section we showed how a controller system for the MQTT protocol is obtained using the FVS framework. We first specified through FVS rules the behavior of the protocol and then a controller is found applying the tools described in Section 2.2

Message Queue Telemetry Transport (MQTT) can be seen as an open-source Machine-to-Machine (M2M) protocol. Communications and interactions between the involved parts (clients and servers/brokers) are carried out employing a classical publish/subscribe architectural pattern. This pattern allows decoupling those processes that generates the information from those processes that receives the information.

The behavior of the protocol is codified employing control packages and three different flavors for Quality of Services are available [4]. In the first one (QoS0), called "at most once" packages can be lost; in the second one (QoS1, called "at least one") there is a guarantee that packages will always arrive but package duplication can occur and finally in QoS2 packages will always arrive without duplication. The desired QoS will depend on the environments conditions where the protocol will be deployed. For example, in most rustic conditions QoS0 can be used and in most critical scenarios where is no place to lose or duplicate packages QoS2 should be employed. When an MQTT connection is made, all the processes will first decide the QoS chosen as a part of the connection setting.

We specified the MQQT behavior in two categories. Rules for managing control packages and rules for the QoS level. In what follows we illustrate some rules in both categories. Rules in Figure 3 deals with the CONNECT package. This packet must be the first one received by the server after the establishment of a network. Additionally, this packet must not be received more than once time. If that happens, a violation of the protocols occurs and the client is disconnected. We use the *OtherMessage* event as a syntactic short-cut to indicate the occurrence of any other event except CONNECT.

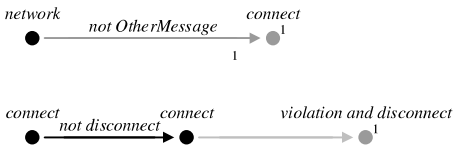

Figure 3. Rules for the CONNECT package.

Figure 4 deals with the acknowledge of the connection package: CONNACK. Either a connection is established or a timeout occurs and the network is closed.

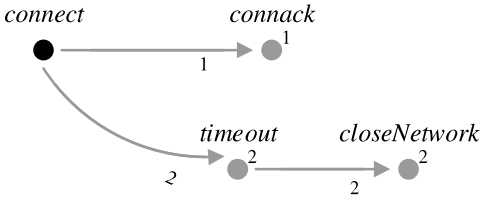

Figure 4. Rules for the CONNACK package.

Similarly, Figure 5 addresses the acknowledge of the PUBLISH package. The PUBACK is the expected response to a PUBLISH package, and after a PUBLISH package the desired QoS is settled.

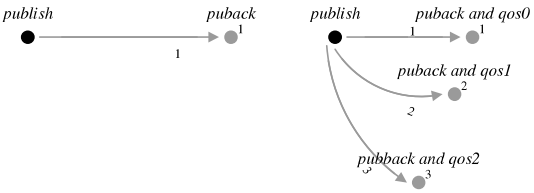

Figure 5. Rules handling the PUBLISH interaction.

In Figure 6 some rules describing typical reply packages are shown. PUBREC is the response to a PUBLISH packet with QoS2, PUBREL is the response to a PUBREC packet, PUBCOMP is the response to a PUBREL packet and finally, PINGRESP is the response to a PINGREQ packet. Also a rule requiring that no messages can be sent after disconnection is also shown in Figure 6.

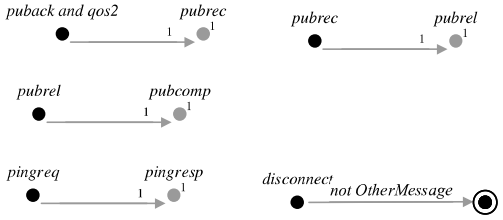

Figure 6. Rules for typical response packages.

The last rules shown in this paper for the package category handles subscriptions to the publish/subscribe buffer (see Figure 7). SUBACK and UNSUBACK packages stand for the acknowledges of the subscription and unsubscribe actions. Also, the level of QoS is defined after a subscription is made.

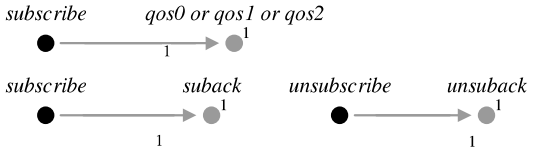

Figure 7. Rules for Subscribing and Unsubscribing.

Finally, FVS rules for managing the desired QoS are exhibited in Figure 8.

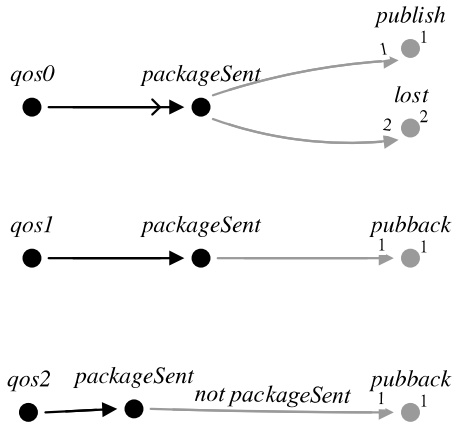

Figure 8. Rules for defining the expected QoS

Some additional rules shaping interactions of the artifacts must also be added. Three examples are shown in Figure 9.

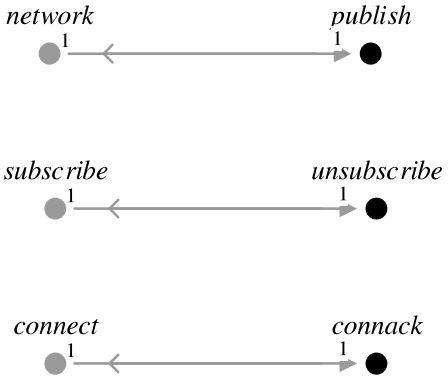

Figure 9. A few extra rules

The rules in Figure 9 say that artifacts cannot publish without establishing a connection (represented by the network event), that artifacts can only unsubscribe if they were subscribed first, and that a CONNACK package con only occur if a CONNECT package occurred previously.

3.1 Obtaining a MQQT controller

With all the rules describing the behavior of the MQTT protocol we obtain a controller using the GOAL tool[16]. Since a controller could be found, then it is guaranteed by construction that it fulfills the specification. Figure 10 shows part the of controller's automaton for the server/broker side.

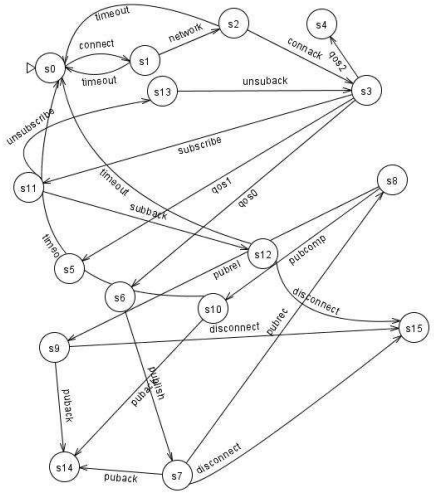

Figure 10. Part of MQQT Server Controller.

Similarly, Figure 11 exhibits part of the behavior of the controller for an MQTT client.

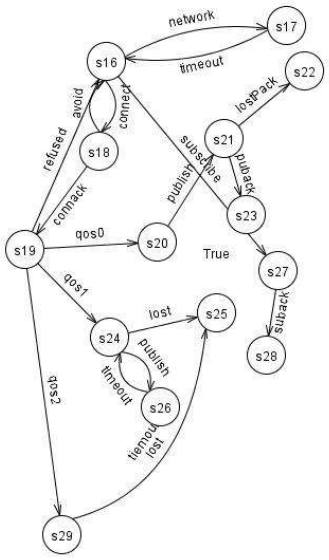

Figure 11. Part of MQQT Client controller.

4. Related Work

Work in [3] presents an appealing survey on formal verification and validation techniques for IoT systems. The techniques are divided into different categories according to the analysis they perform: those analyzing models, those analyzing code, those combining models and code and finally, those performing testing on the systems. In addition to the MQQT protocol, the CoAp protocol is also studied.

Several approaches like [21-23] formally verify IoT protocols focusing on security aspects.

Work in [4] applies model checking techniques to verify the behavior of the MQQT protocol. Two different paths are taken. The first one employs semi-formal models based on UML whereas the second one is based on probabilistic automata.

Research in [5] also formally verified the MQTT protocol using model checkers. Two formalisms are employed: Temporal Logics of Actions [16] and PlusCal [17]. Other interesting works employing model checking are [6-8, 20, 26]. In particular, verification in [26] is achieved through the use of coloured Petri Nets–based models, and a captivating incremental application of model checking levels.

We share with all these approaches the intention to formally verify IoT protocols. However, they are not aimed to automatically obtain a controller like our work. We took into consideration behavior synthesis besides behavioral verification.

Work in [18] also considers synthesis in the IoT Domain. It proposes a novel technique to obtain an intermediary called *Mediator* between all the artifacts in the system to decouple communications and interactions. They are focused in the middleware layer and it is intended for the codification phase. It is based on the Data eXchange (DeX) formalism.

On the contrary, our work proposes a more general scheme, not only for code and not exclusively for middleware. It would be interesting to compare FVS expressive power against DeX to see the power of a potential combination between both approaches.

5. Future Work

Regarding future work we would like to extend our research in several directions.

For one side, we would like to take one step further the controller synthesis approach by employing code generators tools like [13]. This would imply obtaining not only an automaton but also an actual working implementation obtained directly from the specifications.

For the other side, we would like to compare FVS against other approaches like the ones mentioned in Section 4 considering execution time, expressive power and space. This latter aspect could be compared analyzing the size of the automata.

Finally, we would like to extend our empirical validation by adding more case of studies. Our short-term objective is to formally synthesize the behavior of the CoAP protocol.

6. Conclusions

Protocols shape the way different artifacts communicate and interact. They establish how and when information and data must be sent, and by whom. In domains like IoT, where unstable conditions of the environment are usual, verifying that the protocol respect and satisfy its specification is a crucial activity. This is a fertile ground to apply Behavioral Synthesis tools, since the system is built automatically in such a way that the specification is fulfilled by construction.

In this work we introduce the FVS framework as a powerful tool to denote, specify, verify and synthesize behavior in the IoT domain.

In particular, we specified the complete behavior of one of the most used protocols in the IoT world: the MQQT protocol. Additionally, a controller for the system was automatically obtained.

References

[1] Lasi, H., Fettke, P., Kemper, H. G., Feld, T., & Hoffmann, M. (2014). Industry 4.0. Business & information systems engineering, 6(4), 239-242..

[2] Ghavimi, F., & Chen, H. H. (2014). M2M communications in 3GPP LTE/LTE-A networks: Architectures, service requirements, challenges, and applications. IEEE Communications Surveys & Tutorials, 17(2), 525-549..

[3] Hofer-Schmitz, K., & Stojanović, B. (2019, November). Towards formal methods of IoT application layer protocols. In 2019 12th CMI conference on cybersecurity and privacy (CMI) (pp. 1-6). IEEE.

[4] Houimli, M., Kahloul, L., & Benaoun, S. (2017, December). Formal specification, verification and evaluation of the MQTT protocol in the Internet of Things. In 2017 International conference on mathematics and information technology (ICMIT) (pp. 214-221). IEEE.

[5] Shkarupylo, V., Kudermetov, R., Timenko, A., & Polska, O. (2019, October). On the aspects of IoT protocols specification and verification. In 2019 IEEE International Scientific-Practical Conference Problems of Infocommunications, Science and Technology (PIC S&T) (pp. 93-96). IEEE.

[6] J. Hcine and I. B. Hafaiedh, "Formal-based modeling and analysis of a network communication protocol for iot: Mqtt protocol," in International conference on the Sciences of Electronics, Technologies of Information and Telecommunications. Springer, 2018, pp. 350–360.

[7] M. Diwan and M. D'Souza, "A framework for modeling and verifying iot communication protocols," in International Symposium on Dependable Software Engineering: Theories, Tools, and Applications. Springer, 2017, pp. 266–280

[8] 31] A. J. Vattakunnel, N. S. Kumar, and G. S. Kumar, "Modelling and verification of coap over routing layer using spin model checker," Procedia Computer Science, vol. 93, pp. 299 – 308, 2016, proceedings of the 6th International Conference on Advances in Computing and Communications. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1877050916314557

[9] Asarin, E., Maler, O., Pnueli, A., & Sifakis, J. (1998). Controller synthesis for timed automata. IFAC Proceedings Volumes, 31(18), 447-452.

[10] 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).

[11] Baresi, L., Di Nitto, E., & Ghezzi, C. (2006). Toward open-world software: Issues and challenges. Computer, 39(10), 36-43.

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

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

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

[15] 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.

[16] J. E. Johnson, D. E. Langworthy, L. Lamport, and F. H. Vogt, "Formal specification of a web services protocol," Electronic Notes in Theoretical Computer Science, vol. 105, pp. 147–158, December.2004.

[17] L. Lamport, "The PlusCal algorithm language," in 6th Int. Colloquium on Theoretical Aspects of Computing, part of LNCS, Kuala Lumpur, Malaysia, vol. 5684, Aug. 2009, pp. 36–60.

[18] Bouloukakis, G., Georgantas, N., Ntumba, P., & Issarny, V. (2019). Automated synthesis of mediators for middleware-layer protocol interoperability in the IoT. Future Generation Computer Systems, 101, 1271-1294.

[19] 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.

[20] B. Aziz, "A formal model and analysis of an iot protocol," Ad Hoc Networks, vol. 36, pp. 49 – 57, 2016. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S1570870515001183

[21] D. Q. Federico Maggi, Rainer Vosseler, "The fragility of industrial iot's data backbone. security and privacy issues in mqtt and coap protocols," 2018, accessed at: 2019-08-29

[22] L. Nastase, "Security in the internet of things: A survey on application layer protocols," in 2017 21st International Conference on Control Systems and Computer Science (CSCS). IEEE, 2017, pp. 659–666.

[23] S. Arvind and V. A. Narayanan, "An overview of security in coap: Attack and analysis," in 2019 5th International Conference on Advanced Computing & Communication Systems (ICACCS). IEEE, 2019, pp.655–660.

[24] Rabin, M. O. (1963). Probabilistic automata. Information and control, 6(3), 230-245.

[25] Cleaveland, R., & Hennessy, M. (1990). Priorities in process algebras. Information and Computation, 87(1-2), 58-77.

[26] Rodríguez A., Kristensen L.M., Rutle A. (2019) Formal Modelling and Incremental Verification of the MQTT IoT Protocol. In: Koutny M., Pomello L., Kristensen L. (eds) Transactions on Petri Nets and Other Models of Concurrency XIV. Lecture Notes in Computer Science, vol 11790. Springer, Berlin, Heidelberg. https://doi.org/10.1007/978-3-662-60651-3_5

[27] F. Asteasuain, F. Calonge, and M. Dubinsky. Exploring specification pattern based behavioral synthesis with scenario clauses. In CACIC, 2018.

[28] F.Asteasuain, F.Calonge,P.Gamboa , Behavioral Synthesis with Branching Graphical Scenarios. In CONAIISI 2019.

9° CoNaIISI
2021

Congreso Nacional de
Ingeniería Informática y
Sistemas de Información