Vulnerability Identification of Operational Technology Protocol Specifications Through Formal Modeling

Matthew Boeding
Electrical and Computer Engineering
University of Nebraska-Lincoln
Lincoln, NE
mboeding@huskers.unl.edu

Michael Hempel
Electrical and Computer Engineering
University of Nebraska-Lincoln
Lincoln, NE
mhempel@unl.edu

Hamid Sharif
Electrical and Computer Engineering
University of Nebraska-Lincoln
Lincoln, NE
hsharif@unl.edu

***Abstract*—Operational Technology systems are implementing more networking capabilities as increased data visibility, remote management, and automation have become priorities for many system stakeholders. As these previously air-gapped systems become more connected, increased cybersecurity risks are associated with the utilized communication protocols. The energy sector, in particular, relies heavily on a Defense-in-Depth approach to mitigate protocol vulnerabilities that may be exploited by a malicious actor. However, a protocol's specification and any deviations from it for on-device implementations may greatly impact the efficacy of an attack on a device's operation, reliability, or performance. A device with a robust protocol implementation may have lower latency and a higher successful response rate than other devices when under attack. In this paper, we propose the identification of critical transitions in a device's protocol implementation of the Modbus protocol through model-based formal verification utilizing the Construction and Analysis of Distributed Processes toolbox. Following the protocol specification, the models written in the Language Of Temporal Ordering Specification (LOTOS) New Technology (LNT) language were first validated for a single type of packet transaction and subsequently expanded to the full Modbus protocol implementation. The identified critical transitions, in this case, outline any point at which the protocol defines handling unexpected data outside of normal packet processing. We use packet traces captured from real-world infrastructure to validate the correctness of our model in representing an on-device Modbus implementation. Utilizing the identified critical transitions can improve a device's protocol implementation and lessen the impact of a cyberattack on the device or the network.**

***Index Terms*—Modbus, Formal Verification, Construction and Analysis of Distributed Processes, CADP, Smart Grid, Cyber-Physical Systems, Operational Technology**

I. Introduction

The integration and convergence of Information Technology (IT) and Operational Technology (OT) systems has created challenges for securing communications on OT systems [1]. These challenges are specifically of interest to the energy sector, where the proliferation of Distributed Energy Resources (DERs) necessitates an associated increase in communications capabilities. However, currently implemented protocols were not designed with security in mind, and the requirement for constant availability often discourages adding additional security measures to the protocols. As such, a Defense-in-Depth approach is employed to avoid system-wide outages caused by cyberattacks [2]. The approach often entails adding network monitoring tools such as firewalls, intrusion detection systems, and anomaly detection schemes. While these methods improve overall cybersecurity posture, the protocols are still vulnerable to various attacks [3]

The Modbus [4] protocol has been widely adopted for use within Industrial Control Systems (ICS). While the protocol specification outlines the packet formation and exception handling requirements, the device-specific implementation requirements are not. This allows for significantly different performance and behavior of devices with seemingly identical protocol capabilities. The current method of identifying these device differences and interoperability is completed through the creation of testbeds, software in the loop (SIL), hardware in the loop (HIL) simulators, or digital twins [5]. There is currently no standard to identify the causes of diverging performance due to differences between protocol implementations. Thus, we describe our method to identify critical transitions within protocols and their implementations to enable improved robustness and compliance through formal modeling.

The remainder of the paper is structured as follows. Section II introduces related works for evaluating protocol implementations and previous formal modeling works. Section III provides an overview of our methodology for translating protocols into a formal model and the implementation, including the chosen formal modeling software. Section IV provides an overview of the achieved model, including the number of states in the state machine, reduction techniques, and identified critical transitions of the model. Section V concludes the paper and outlines our future work related to this work.

II. Related Works

Formal verification methods have been employed on ICS protocols in prior efforts. In [6], the authors utilized formal verification to evaluate the security vulnerabilities of the Modbus Protocol by adding a malicious actor to intercept traffic through a TCP/IP switch. Formal verification was also

employed to examine the security of DNP3's secure authentication method in [7], which follows a specific handshake procedure similar to TLS. A formal TLS handshake model was also shown in [8], which could find multiple incorrect handshake scenarios. Formal methods are employed in other fields and have been shown to work in modeling concurrent asynchronous systems such as for the smart grid [9], based on the Construction of Distributed Processes (CADP) toolbox [10]. Other tools, such as TLA+ [11], can be used to design models for safety-critical aerospace systems [12], and Register Transfer Logic models [13] based on commercial tools. There appear to be only a limited number of formal methods employed with regard to OT systems. One cause for this limit is the ability to test individual system configurations with Digital Twins [5]. However, this method is ineffective when identifying issues with increasingly complex systems. Previous research has also outlined the difficulty of implementing formal methods compared to other methods and the required domain knowledge and formal method experts [14].

III. Methodology

From the variety of applications that formal methods have been applied to, we propose implementing ICS protocol models for formal verification and analysis using the CADP toolbox. CADP has been widely used for process algebra and formal modeling of concurrent systems [15]. This paper aims to create a formal model of the Modbus protocol and identify any critical transitions that may be affected by on-device implementation deviations. We identify critical transitions as any point where a protocol exhibits nondeterministic behavior, i.e., a timeout to return to an idle state. Although each device must handle this state, their implementation may be different and cause, for example, increased latency for even low-volume attack traffic or device failure.

Our previous work demonstrated this effect by using our IEC-61850 Generic Object Oriented Substation Event (GOOSE) protocol testbed [16]. A device under test showed a latency increase from 10 ms to 480 ms as soon as attack traffic was applied to the device, compared to a second device that kept latency close to the baseline 10 ms, shown in Figure 1.

The process we follow to create the formal model for the Modbus protocol includes translating the protocol specification to a hierarchical diagram and generating Lotos New Technology (LNT) specification language [17]. Translating the protocol specification to a hierarchical diagram includes identifying communication channels, individual protocol node states, and packet types supported by the protocol. In the case of the Modbus protocol, there are two nodes we must consider: a client and a server. These nodes perform a request/response transaction for each packet. LNT utilizes rendezvous synchronization communication processes, making it well-suited for modeling communication protocols. ICS devices can have multiple different processes operating concurrently but are still required to communicate when a packet or synchronization event occurs.

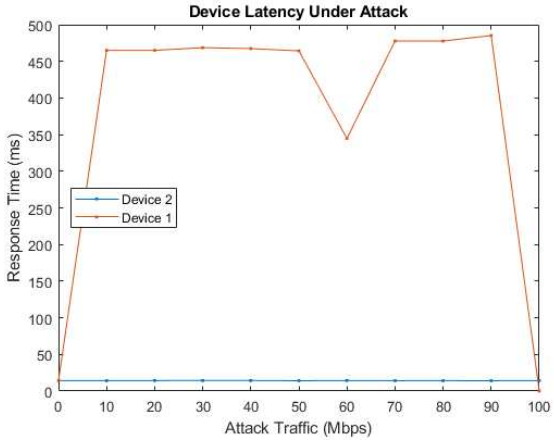

Fig. 1. Device Response Under Attack [16]

A. Protocol Translation

As a communication method, the Modbus protocol outlines two separate nodes for communication, the client and the server. The communication method between these two nodes is request/response, so the client will not transition from an idle state until it is ready to send a packet and a packet is received on the server side. Figure 2 shows the state machine for each node. Modbus is effectively a stateless protocol. Each transaction is embodied by a request that is concluded when the corresponding response is received or the exchange times out. Effectively, the validation of all aspects related to a single request-response message exchange also constitutes the validation of the entire protocol activity, as shown in [18]. Specifically applied to our work, it also implies that modeling a single transaction will identify all possible critical states within the protocol specification.

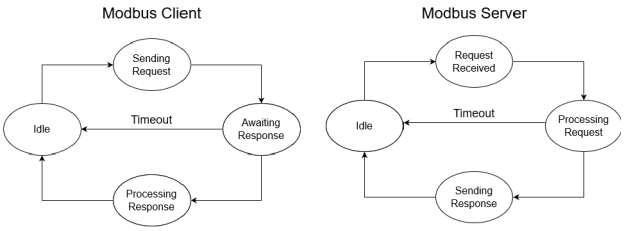

Fig. 2. Modbus Device States

The packets within each defined function code have separate packet structures, server processing steps, and response packet structures. To translate the protocol into a formal model understanding, the packet structure for each function code was organized and represented as a hierarchical structure. In total, the Modbus protocol has 19 defined function codes, and this process was completed for each request and response function code. An example illustrating the Read Coils request packet hierarchy is shown in Figure 3.

The packets outlined here quickly lead to the issue of an exploding state space, as each packet field has a large

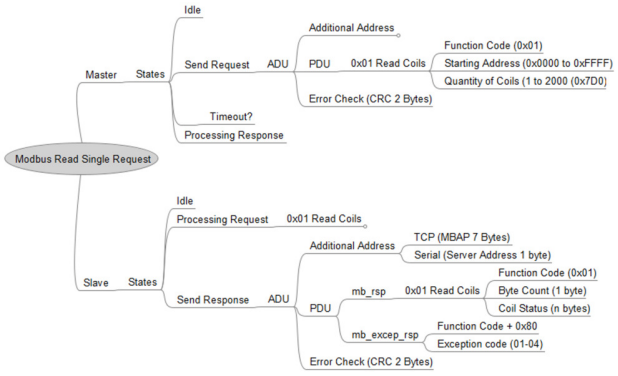

Fig. 3. Read Coils Hierarchical Representation

number of values. For example, a simple starting address field accepting values in the range of 0x0000-0xFFFF results in 65,536 states for this field alone. If we were to generate a model within LNT with the current packet fields, the model would find exponentially more states than those required to maintain the protocol implementation. To combat this, we collapse the value ranges into binary decisions, as outlined in the next section of this paper. Additional simplifications can be applied, such as for the additional address field, identified as MBAP for TCP/IP and as Server Address for serial communications, which can be treated as the same field due to the fact that the protocol does not specifically outline processing of this field for the different physical media.

B. Formal Model

Once the protocol is translated into an understandable diagram, the model must be translated into LNT for validation. In this case, we utilize the nondeterministic finite automata (NFA) outlined in [19], to represent the protocol as the tuple $(Q, \sum, \delta, q_i, F)$. Their relation can be seen in Table I. Since the Modbus protocol does not have a set packet format for all available function codes, each packet request and response have to be evaluated individually. In section IV, the process will be outlined for an individual packet, then translated to the entire protocol.

TABLE I
NFA TO PROTOCOL TRANSLATION

Element	NFA Element	Protocol Element
Q	Set of States	Modbus Device States
$\sum$	Alphabet	Modbus Packet
$\delta \subseteq Q$ x $\sum$ x Q	Transition Relation	Packet Sent Over Wire
$q_I \in Q$	Initial State	Start of Transaction
$F \subseteq Q$	Final State	End of Transaction

When translating to LNT, processes, packets and packet fields were individually defined. The initial processes are broken down into both Client and Server. These processes then call child processes for each notable device state, which include sending request, request received, or sending response shown in Figure 2. In this example model, the process assumes ideal physical channel conditions, so no loss of packets from the transmitter to the receiver is considered. The communication channel accepts both requests and responses, and timeouts are used for recovering from invalid packets, as outlined in the protocol specification.

```
process Main
    [physical:Physical, timeout:Timeout]
    is
     par
          Client[physical, timeout]
          ||
          Server[physical, timeout]
    end par
end process
```

Each packet field was reorganized to contain a binary decision based on the required packet processing outlined within the protocol specification. For example, the server address, starting address, the quantity of coils, and the error-checking fields can have two possible values. In this way, a Read Coils request with a server address, function code, starting address, quantity of coils, and error checking field can be reduced from 1.09e+15 possible values to 2^4 values:

```
type ServerAddress is
    CORRECT,
    INCORRECT
    with ==
end type

type InternalAddress is
    VALID,
    INVALID
    with ==
end type

type InternalQuantity is
    VALID,
    INVALID
    with ==
end type

type CRC is
    CORRECT,
    INCORRECT
    with ==
end type
```

IV. Model Creation Results

The model creation within CADP was executed in two stages; the first was creating a Read Coils model, utilizing only one function code for transaction validation. Strong equivalence reduction was then used to find a minimal form of the Read Coils transaction. Next, we identified Critical transitions, shown as hidden label *"i"* in the model, and examined them for critical implementation states of the protocol. There are

also more severe issues that can be identified using CADP, including deadlocks and livelocks. These states occur when different processes attempt to continue operating in divergent states. Deadlocks identify that multiple members are stuck at a particular state, each waiting for the other to generate an event needed to transition to another state. Livelocks identify a loop of states that cannot be exited.

Compilation of each model was completed utilizing Script Verification Language (SVL) [20] to generate Binary Coded Graphs for state space exploration. Using the binary type definitions shown in Section III, the Read Coils transaction was compiled to have 18 states, with 92 transitions and 12 transitions with the hidden label *"i"*. A visual representation of this model can be seen in Figure 4. Upon investigation of the model, no deadlocks or livelocks were found within this implementation.

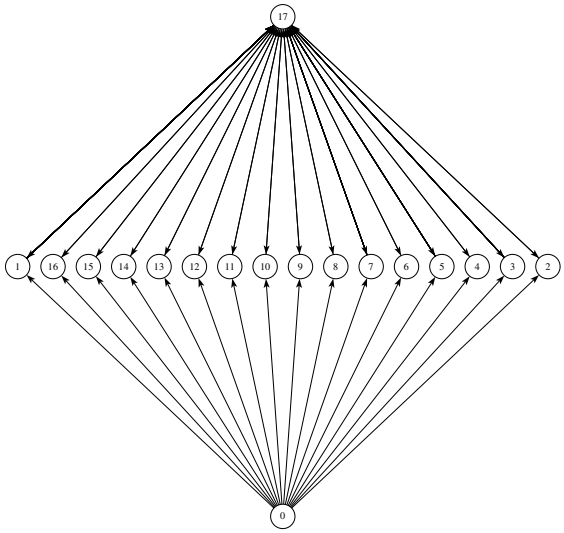

Fig. 4. Modbus Read Coils Single Transaction

This model may be reasonable to examine visually for critical transitions. However, the entire protocol has 19 function codes, with request, response, and exception packets available for each transaction. For this reason, branching reduction was completed utilizing the SVL language to assist with the visual identification of individual states. The reduced model size reduces the transaction to 4 unique states, with 56 transitions, shown in Figure 5. The reduced model shows that any transition from state 4 to state 0 will constitute a critical transition. A drawback of using this reduction is the removal of hidden labels for timeouts that identify critical transitions. Since this is the primary interest of the model, we can utilize OPEN/CAESAR [21] tool included within the CADP toolbox to visually find the critical transitions, as shown in Figure 6.

These identified states can be summarized as any state where the server address or error check field is incorrect. There are 16 total packet combinations for the Read Coil exchange, but there will only be 4 possible combinations where the

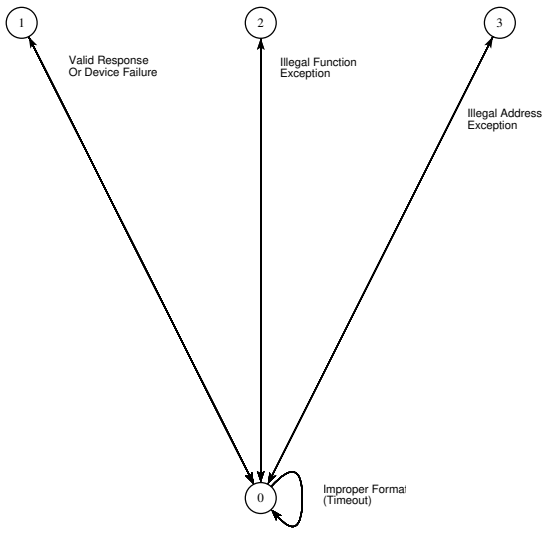

Fig. 5. Modbus Read Coils Reduced

only changing fields are starting address or value. Upon first examination, it may seem odd that the response packet has no effect on the critical transitions of the protocol. However, the client and server are both reliant on processing the request packet, and the client may request the same data later with no adverse effect on the overall system performance. The server examines the other fields in each packet, and individual exception packets are sent for each incorrect value. In the case of Read Coils, there are exceptions for unsupported function code, invalid addresses, and device failures.

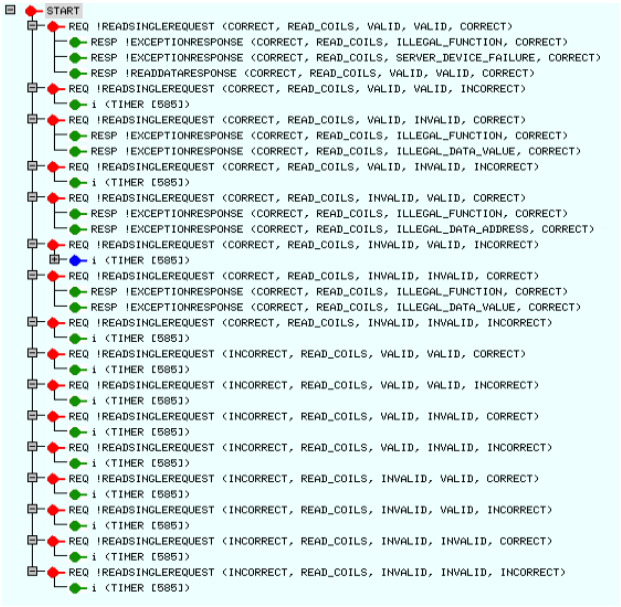

Fig. 6. Examination of critical transitions

Once the individual packet model was completed, the proto-

col specification was written with LNT. Examination of each individual packet was completed. Across all 19 function codes we identified 12 unique packet structures for requests, 14 for responses, and 1 for exception codes. Packet similarities were identified by packet field, as Read Coils and Read Register may have differing cut-off values for a valid address or quantity, the packet field can be translated to either a valid or invalid address. To encompass the whole protocol, the physical channel for communication was defined to accept 27 different packet types, each of which is composed to match the protocol specification. The model was allowed to decide at runtime on any specific packet for transmission. Definitions for each applicable packet field were defined with LNT keyword *any*, to encompass all possibilities of a single protocol transaction between client and server.

Once the communication channel, packet types, and processing stages were completed for the protocol, the generation of the model was completed in the same manner as the single-packet model. Compilation of this model created a 787-state model with 11,385 transitions and 5,799 critical transitions with the hidden label *"i"*. Reduction of this model creates a 62-state model, which can be seen in Figure 7. From the state machine's visualization, all transitions with start and endpoint 0 constitute a critical transition. The model contained no deadlocks or livelocks across all packet transactions, implying the protocol specification has considered appropriate timeout rules for each device. Examination of the critical transitions for the protocol identified the same required fields as the server address and error-checking field. These fields are not accounted for in packet processing. However, they may be received by a Modbus server regardless of serial or TCP/IP implementation.

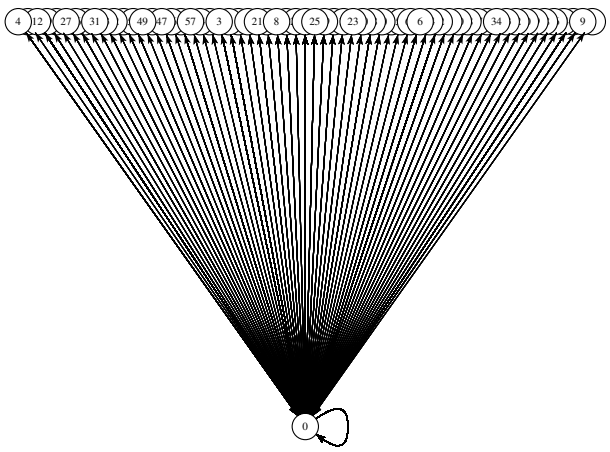

Fig. 7. Modbus Full Protocol Implementation

The final results of the model generation and comparison can be seen in Table II. Even though Modbus is a widely adopted protocol due to its open standard and simplicity, many critical transitions can still negatively impact device implementations, especially during cyberattacks. These transitions can be especially difficult to identify in distributed systems, as each device's requirements and target applications can differ significantly. Additionally, given the fact that protocol implementations can also vary extensively for different vendors, or even between different products of the same vendor, the identification of critical transitions, deadlocks, or livelocks is vital to creating a more robust and compliant protocol ecosystem.

TABLE II
MODBUS CRITICAL TRANSITION AND REDUCTION RESULTS

Model	States	Transitions	Critical Transitions	Reduced States
Read Coils	18	92	16	4
Full	787	11,385	5,799	62

V. CONCLUSION AND FUTURE WORK

Traditionally designed for air-gapped serial wire implementations, OT protocols are rapidly transitioning towards greater interconnectivity and automation capabilities, often at the expense of the cybersecurity provided to a device. This convergence of OT and IT systems has created a greater attack surface for malicious actors. These systems often require reliance on Defense-in-Depth strategies to improve the security of a network, but more comprehensive actions are required. One such endeavor is ensuring the robust and compliant implementation of protocols, which can significantly help mitigate the effects of cyberattacks on a device.

In this paper, we introduced a method for identifying vulnerabilities of OT protocols within device implementations utilizing formal modeling and the CADP toolbox. We outlined the translation of the Modbus protocol to LNT utilizing binary decisions for packet fields and the model size reduction through various methods. By utilizing formal modeling to identify critical transitions within a protocol, we can create guidelines for more robust protocol implementations to alleviate the current burden placed on Defense-in-Depth in ICS networks. Our future work will expand this method to additional OT protocols, including the DNP3 and IEC-61850 GOOSE protocols. These protocols and their analysis will demonstrate the great flexibility inherent to our approach, as they implement a single packet structure, but have more complex processing and assembly compared to the Modbus protocol demonstrated here. Thus, rather than having a variety of packet types with two packet fields that determine critical transitions, there will be a single packet type with multiple fields determining critical transitions. We use packet traces captured from real-world infrastructure to validate the correctness of our model in representing an on-device Modbus implementation. This work and its findings were validated against real-world Modbus packet traces captured on actual Modbus infrastructure.

ACKNOWLEDGMENT

This research has been supported in part by the University of Nebraska-Lincoln's Nebraska Center for Energy Sciences

Research (NCESR) under Cycle 16 Grant# 20-706.

REFERENCES

[1] P. K. Garimella, "IT-OT Integration Challenges in Utilities," in *2018 IEEE 3rd International Conference on Computing, Communication and Security (ICCCS)*, 2018, pp. 199–204.

[2] K. Khanna, G. Ravikumar, and M. Govindarasu, "Defense-in-Depth Framework for Power Transmission System against Cyber-Induced Substation Outages," in *2023 IEEE Texas Power and Energy Conference (TPEC)*, 2023, pp. 1–6.

[3] M. Boeding, K. Boswell, M. Hempel, H. Sharif, J. Lopez, and K. Perumalla, "Survey of Cybersecurity Governance, Threats, and Countermeasures for the Power Grid," *Energies*, vol. 15, no. 22, 2022. [Online]. Available: https://www.mdpi.com/1996-1073/15/22/8692

[4] "MODBUS Application Protocol Specification v1.1b3," *MODICON Inc. Industrial Automation Systems Tech. Rep*, 2012.

[5] A. Löcklin, M. Müller, T. Jung, N. Jazdi, D. White, and M. Weyrich, "Digital Twin for Verification and Validation of Industrial Automation Systems – a Survey," in *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, vol. 1, 2020, pp. 851–858.

[6] R. Nardone, R. J. Rodríguez, and S. Marrone, "Formal security assessment of Modbus protocol," in *2016 11th International Conference for Internet Technology and Secured Transactions (ICITST)*, 2016, pp. 142–147.

[7] R. Amoah, S. Camtepe, and E. Foo, "Formal modelling and analysis of DNP3 secure authentication," *Journal of Network and Computer Applications*, vol. 59, pp. 345–360, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1084804515001228

[8] J. Bozic, L. Marsso, R. Mateescu, and F. Wotawa, "A Formal TLS Handshake Model in LNT, journal = Electronic Proceedings in Theoretical Computer Science," vol. 268, pp. 1–40, mar 2018. [Online]. Available: https://doi.org/10.4204%2Feptcs.268.1

[9] K. Hafdi and A. Kriouile, "Formal Modeling and Validation of Micro Smart Grids Based on ReDy Architecture," in *2020 5th International Conference on Cloud Computing and Artificial Intelligence: Technologies and Applications (CloudTech)*, 2020, pp. 1–7.

[10] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "CADP 2011: A Toolbox for the Construction and Analysis of Distributed Processes," *International Journal on Software Tools for Technology Transfer*, vol. 15, pp. 89–107, 03 2013.

[11] L. Lamport, *Specifying Systems: The TLA+ Language and Tools for Hardware and Software Engineers*. USA: Addison-Wesley Longman Publishing Co., Inc., 2002.

[12] S. Paul, E. Cruz, A. Dutta, A. Bhaumik, E. Blasch, G. Agha, S. Patterson, F. Kopsaftopoulos, and C. Varela, "Formal Verification of Safety-Critical Aerospace Systems," *IEEE Aerospace and Electronic Systems Magazine*, pp. 1–14, 2023.

[13] M. Girish, G. Gopakumar, and D. S. Divya, "Formal and Simulation Verification: Comparing and Contrasting the two Verification Approaches," in *2021 2nd International Conference on Advances in Computing, Communication, Embedded and Secure Systems (ACCESS)*, 2021, pp. 41–44.

[14] A. Naumchev, A. Sadovykh, and V. Ivanov, "VERCORS: Hardware and Software Complex for Intelligent Round-Trip Formalized Verification of Dependable Cyber-Physical Systems in a Digital Twin Environment (Position Paper)," in *Software Technology: Methods and Tools*, M. Mazzara, J.-M. Bruel, B. Meyer, and A. Petrenko, Eds. Cham: Springer International Publishing, 2019, pp. 351–363.

[15] H. Garavel, F. Lang, R. Mateescu, and W. Serwe, "Is CADP an Applicable Formal Method?" *Electronic Proceedings in Theoretical Computer Science*, vol. 349, pp. 1–11, 11 2021.

[16] M. Boeding, M. Hempel, H. Sharif, J. Lopez, and K. Perumalla, "A flexible ot testbed for evaluating on-device implementations of iec-61850 goose," *International Journal of Critical Infrastructure Protection*, p. 100618, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1874548223000318

[17] D. Champelovier, X. Clerc, H. Garavel, Y. Guerte, F. Lang, C. McKinty, V. Powazny, W. Serwe, and G. Smeding, "Reference Manual of the LOTOS NT to LOTOS Translator Version 7.2," Mar 2023.

[18] J. M. Taylor and H. R. Sharif, "Enhancing Integrity of Modbus TCP Through Covert Channels," in *2017 11th International Conference on Signal Processing and Communication Systems (ICSPCS)*, 2017, pp. 1–6.

[19] R. Cleaveland, *Better Automata Through Process Algebra*. Cham: Springer Nature Switzerland, 2022, pp. 116–136. [Online]. Available: https://doi.org/10.1007/978-3-031-15629-8_7

[20] H. Garavel and F. Lang, "SVL: A Scripting Language for Compositional Verification," in *Formal Techniques for Networked and Distributed Systems*, M. Kim, B. Chin, S. Kang, and D. Lee, Eds. Boston, MA: Springer US, 2001, pp. 377–392.

[21] H. Garavel, "OPEN/CÆSAR: An open software architecture for verification, simulation, and testing," in *Tools and Algorithms for the Construction and Analysis of Systems*, B. Steffen, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 68–84.