A Security Testbed for Networked DES Control Systems ⋆

Michel R. C. Alves * **Karen Rudie** ** **Patrícia N. Pena** ***

* *Graduate Program in Electrical Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil (email: michelrodrigo@ufmg.br)*
** *Department of Electrical and Computer Engineering, and Ingenuity Labs Research Institute, Queen's university, Kingston, Ontario, K7L 3N6, Canada (email: karen.rudie@queensu.ca)*
*** *Department of Electronics Engineering - Universidade Federal de Minas Gerais - Av. Antônio Carlos 6627, 31270-901, Belo Horizonte, MG, Brazil (email: ppena@ufmg.br)*

Abstract: In this paper we present an implementation scheme for a networked control system where the coordination between devices is done by supervisory control theory. The system is intended to be used as a testbed for DES security-related techniques. We provide some examples of attacks whose effect can be evaluated using the proposed setup. The hardware and software are based on low-cost devices and can be modified to suit different control systems. The details of the implementation are available on a free online repository.

Keywords: Discrete-Event Systems - Supervisory control - Deception attacks - Networked Control

1. INTRODUCTION

To be competitive in today's market-driven economy, organizations employ efficient industrial control systems (ICS) that rely on network communicating devices. However, the increased efficiency that ICS introduce also presents new security issues, a fact that is drawing the attention of many researchers. Attacks can happen at different levels in an ICS, each of which determines the targeted information and impact on the system (Mourtzis et al., 2019). Because of the nature of the control of systems modeled as discrete-event systems (DES), which is the coordination between subsystems while specifications are met, it is natural for for the control to be located at levels close to the shop-floor within an ICS. This is the kind of system investigated in this paper.

Most attacks on ICS can be classified either as deception attacks or DoS (Denial of Service) attacks, which compromise data integrity and data availability, respectively (Yuan et al., 2019). Normally, the interfaces between devices and networks have known vulnerabilities that are targets for malicious agents to perform deception attacks (Pan et al., 2020; Mourtzis et al., 2019). As pointed out by Rashidinejad et al. (2019), the approaches that deal with security in DES can be classified mainly according to three aspects: *i)* attack location; *ii)* attack impact on transmitted data and; *iii)* security mechanism. Regarding the attack location, most approaches developed by the DES community consider attacks in the communication channel from the plant to the controller, in the communication channel from the controller to the plant or in both channels. This is different from what the literature on cyber-attacks describe as attack locations, since there is no division between the communication channels in practice. Because the techniques for defense or attack design proposed in the DES literature depend heavily on the system and attack modeling, we perceive an inconsistency between the theoretical and real worlds.

Aiming to reduce the gap between theory and practice, in this work we present an implementation scheme to be used as testbed for attack design and defense techniques. It is not the intent of this paper to propose any method for attack design or to provide a defense strategy; we only propose a prototype where such techniques can be tested. This is done by employing low-cost off-the-shelf microcontrolled devices to implement the control of a hybrid networked system. The physical system is simulated in hardware, allowing easy modifications on the control system to fit any other industrial process with the advantage of having all the DES-related control running as it would be if the physical system were also implemented. Furthermore, one can connect a real process to the control system if desired and we provide detailed information about the hardware and software in a free online repository.

This paper is organized as follows. In Section 2 we present some basic preliminaries about automata and supervisory control theory. In Section 3 we present the industrial process in which the implementation proposed is based on. Section 4 discusses the possibilities of attacks on such system. In Section 5, we present the proposed implementation scheme, detailing its architecture, hardware and

⋆ This work has been supported by the National Council for Scientific and Technological Development - CNPq under grant 443656/2018-5, Brazil, and the Natural Sciences and Engineering Research Council of Canada - NSERC.

software. In Section 6 we make some considerations about the proposed solution and finally, in Section 7, we present our conclusions and directions for future work.

2. PRELIMINARIES

The behavior of a DES is modeled by strings of symbols from an alphabet Σ. The Kleene closure Σ^* is the set of all strings over Σ, including the empty string ε. Consider strings $s, v, t \in \Sigma^*$. The concatenation of s with v forms the string $t = sv$ and we can say that s is a prefix of t. Any subset $L \subseteq \Sigma^*$ is called a *language*. The *prefix closure* $\overline{L}$ of L is the set of all prefixes of strings in L.

An automaton is defined as a tuple $G := (Q, \Sigma, \delta, q_0, Q_m)$, where Q is a finite set of states, Σ is the nonempty finite set of symbols called the alphabet, $\delta : Q \times \Sigma \to Q$ is a partially defined transition function, q_0 is the initial state and $Q_m \subseteq Q$ is a set of marked states, which normally represent the completion of tasks. The notation $\delta(q, \sigma)!$ indicates that $\delta(q, \sigma)$ is defined for some $q \in Q$ and $\sigma \in \Sigma$. The transition function δ can be extended to a function $Q \times \Sigma^* \to Q$ according to $\delta(q, \varepsilon) := q$ and $\delta(q, s\sigma) := \delta(\delta(q, s), \sigma)$, with $q \in Q$, $s \in \Sigma^*$, $\sigma \in \Sigma$. The map $\Gamma : Q \to 2^{\Sigma}$ defined as $\Gamma(q) := \{\sigma \in \Sigma | \delta(q, \sigma)!\}$ is the set of *feasible events* at state $q \in Q$. The language generated and marked by G, denoted by $\mathcal{L}(G)$ and $\mathcal{L}_m(G)$, respectively, are defined as $\mathcal{L}(G) := \{s \in \Sigma^* | \delta(q_0, s)!\}$ and $\mathcal{L}_m(G) := \{s \in \mathcal{L}(G) | \delta(q_0, s) \in Q_m\}$.

Supervisory control theory (SCT) is a formal method, based on languages and automata theory, to solve the problem of controlling discrete-event systems. The system to be controlled is called the *plant*, the controller agent is called the *supervisor* and the control problem is to find a supervisor which enforces the specifications in a minimally restrictive way. The uncontrolled behavior of the plant is modeled as an automaton G where transitions are associated with symbols in Σ. Some of the events can be disabled by an external agent and are called *controllable*, represented by Σ_c. The events that cannot be prevented from happening are called *uncontrollable*, denoted by Σ_u and $\Sigma = \Sigma_c \dot{\cup} \Sigma_u$. The role of the supervisor S is to regulate the plant's behavior to meet a desired behavior K by disabling controllable events. Let E be an automaton that represents the specification imposed on G. We say that $K = \mathcal{L}_m(G||E)$, where $||$ represents the parallel composition operation, which synchronizes the automata on the common events while allowing unrestricted execution of private events. The generated and marked languages of the closed loop system are represented by $\mathcal{L}(S/G)$ and $\mathcal{L}_m(S/G)$, respectively. The necessary and sufficient condition for the existence of a nonblocking supervisor S for G such that $\mathcal{L}_m(S/G) = \mathcal{L}_m(G||E) = K$ is that K has to be *controllable* with respect to $\mathcal{L}(G)$ and Σ_u, namely, $\overline{K}\Sigma_u \cap \mathcal{L}(G) \subseteq \overline{K}$. If K is not controllable, then a supervisor is obtained by taking the supremal controllable sublanguage of K, denoted by $\sup \mathcal{C}(K, G)$. This language always exists.

3. THE PROCESS

In this section we describe the physical process in which we applied the proposed control solution. Although its components and models are already defined, they could easily be adapted in order to model any other real process. The general idea is the production of batches of a liquid that is the product of reactions that happen at different temperatures inside a reactor. A depiction of the process can be seen in Fig. 1.

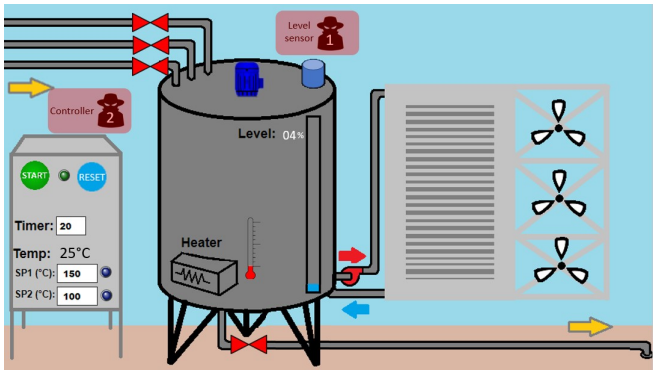


Fig. 1. Representation of the process. The red rectangles represent the presence of a malicious agent.

The process starts with the admission of three different liquids, after the start button is pressed by an operator. It is assumed that the flow of each liquid is defined by a previous process. There are three on-off input valves, each one for a different liquid, that receive the same control action, that is, they open and close together. Since the valves remain open for the same amount of time, the proportion of each liquid in the tank is determined by their flow.

The total amount of liquid in the tank, is a controlled variable and it is measured by a level sensor. Once the level in the tank reaches the setpoint, the input valves are closed and the mixer is turned on. The mixer makes the mix more homogeneous and therefore diminishes the total time of the batch production.

A heating control system is also in place and after the mixer has started, a continuous-time PI controller makes the temperature reach a desired value and remain there for a predefined period of time. After that, the controller makes the temperature reach a second predefined value, which is lower than the previous one, for another amount of time. A heater installed in the tank makes it possible to increase the temperature. To allow the liquid to cool faster, there is a heat exchanger coupled to the tank. To make the liquid go through it, a pump is turned on. Once this part of the process is finished, the mixer and the pump stop and an output valve is opened. The valve will remain open until the level sensor indicates that the tank has reached the low level. When this happens, the output valve is closed and the batch is finished. The system is now ready to produce another batch.

4. ATTACKS ON CONTROL SYSTEMS

In this paper, we focus on deception attacks in the control of DES. Due to the heterogeneity of systems and attack models found in the literature (Rashidinejad et al., 2019), it is not straightforward to make a comparison between different prospective approaches. To establish a common ground for testing such techniques we propose a testbed

based on real control systems. By doing so, we bring to attention many of the aspects that are inherent to real applications and which have to be addressed properly. Additionally, the proposed testbed is flexible enough to accommodate techniques that vary on attack location, attack impact on transmitted data and security mechanism. Example 1 illustrates attacks on different locations.

Example 1. Consider the system shown in Fig. 1. A malicious agent can infiltrate the level sensor in the tank, represented by the red rectangle at the top of Fig. 1. At that position, it can alter the information coming out of the sensor or the information being received by it, by inserting or erasing events. A second possibility would be the one where the attacker has gained access to the system controller, represented by the red rectangle on the left of Fig. 1. In such a case, the attacker is able to modify (i.e., by inserting or erasing events) all the information being received or sent by the controller. □

Note that in the proposed testbed, we assume that the attacker has already infiltrated the system. How this is done by the attacker is outside the scope of this work. The proposed solution offers access to the incoming and outgoing communication of the devices in the system, allowing an attack function provided by the user to insert or erase events. An attack function describes which set of actions can be taken by the attacker at the current state of the system. Thus, different kinds of deception attacks can be reproduced and their impact on the system analyzed.

In order to detect attacks, there is a special device, called the intrusion detection system (IDS). The IDS knows the models of all subsystems and supervisors. It observes all the events exchanged by the devices and updates its models accordingly. If it detects a behavior that is not legal, then it triggers an alarm. We assume that the IDS is immune to attacks. Note that we do not consider the IDS as a defense mechanism. It is only an obstacle that an attacker has to overcome if it wants to remain hidden. If desired, the IDS can be removed or replaced by another detection mechanism.

Regarding the actions of the attacker, we allow it to insert or erase events. However, we can further detail these actions by analyzing *where* the events are being inserted to or removed from. Consider Fig. 2. It shows a controller and subsystems connected in a bus network. Each device has a network interface, which is represented by the inner dotted rectangle. Once the attacker has infiltrated the device, it has five different possible actions, represented by different letters in Fig. 2:

- *a* - The attacker does not interfere with the communication
- *b* - The attacker inserts an event into the network;
- *c* - The attacker inserts an event into the device observation;
- *d* - The attacker erases an event that arrived through the network;
- *e* - The attacker erases an event that was sent by the device.

In order to allow an attacker to perform the actions shown in Fig. 2, the testbed offers access to the incoming events before they are seen by the device and to the outgoing

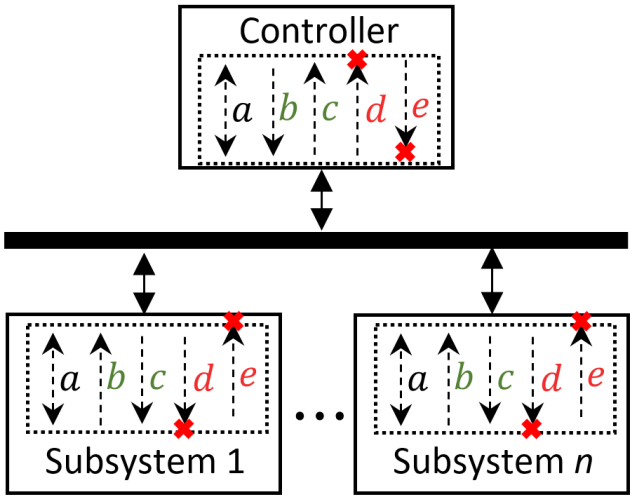
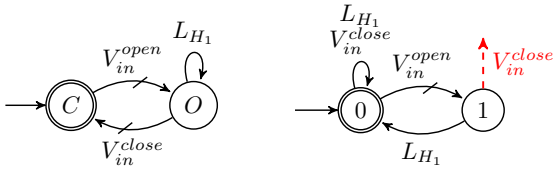


Fig. 2. Possible attacker actions.

events before they are sent to the communication network. Note that these actions can be combined to represent more complex attack behavior. Furthermore, even if in the real attack the attacker does not act in the network interface, the testbed allows one to mimic the behavior of such an attack and to analyze its impact over the control system. Next we present Example 2, that shows the impact of the attackers action according to the attack location.

Example 2. Consider the model of the input valve V_{in}, shown in Fig. 3(a), and the supervisor S_1, shown in Fig. 3(b). The valve is initially closed. Event V_{in}^{open} opens the valve and event V_{in}^{close} closes it. While the valve is opened, the level sensor can trigger event L_{H_1}, which indicates that the tank is full. The supervisor S_1 controls the closing of the valve, by enabling event V_{in}^{close} only after the tank is full. Events V_{in}^{open} and V_{in}^{close} are controllable while event L_{H_1} is uncontrollable.



(a) Input valve - V_{in} (b) Reduced supervisor S_1

Fig. 3. Model of the input valve V_{in} and reduced supervisor S_1. Dashed red arrows represent disabled events.

The internal structure of the IDS will have the model of the plant and a reduced supervisor. This allows the IDS to know the closed-loop behavior of the system, given by $\mathcal{L}_m(V_{in}||S_1)$.

Now suppose we want to study the impact of an attacker that has infiltrated the level sensor and is able to insert or erase events. Consider that event V_{in}^{open} happened and the tank is being filled. If the attacker knows the system, then two possible attacks would be:

(1) Insert event L_{H_1} into the communication network before the level reaches the setpoint. Note that this event has not occurred yet in the plant but it was sent into the network by the attacker;

(2) Erase event L_{H_1} when it happens, preventing it from reaching the network, and insert it into the communication network later.

In both cases, the behavior as seen by the IDS is still legal. For case (1) the attack would cause the input valve to be closed earlier than it should, resulting in the production of a lower quality batch. For case (2), the attack can cause overflow in the tank, which may draw the attention of a human operator.

Consider now that the attacker is able to infiltrate the controller and is able to insert or erase events. Again, consider that event V_{in}^{open} happened and the tank is being filled. Two possible attacks would be:

(3) Insert event L_{H_1} into the controller's observation (which will result in the insertion of the event into the supervisor's observation) before the level reaches the setpoint.Note that the inserted event never reaches the network;
(4) Erase event L_{H_1} when it arrives at the controller and then insert it into the controller's observation later.

Now, case (3) will trigger an alarm in the IDS, since the controller, after seeing event L_{H_1} that was inserted by the attacker, will trigger event V_{in}^{close}. However, the IDS, which only tracks events transmitted in the network, will only see the occurrence of events V_{in}^{open} and V_{in}^{close}, which is not a legal behavior. Note that the IDS and the controller have different observations when the attacker is in the controller. In case (4), although the tank may overflow, the attack will not trigger an alarm in the IDS, since it will see a correct sequence of events. □

In Example 2, one can conclude that the attacker is successful in cases 1), 2) and 4), since it caused a misbehavior in the system and remained undetected. This highlights the fact that the proposed IDS is not a good defense strategy, justifying the development of other defense mechanisms by the user.

In the next section we present some aspects regarding the implementation of the proposed testbed.

5. IMPLEMENTATION OF THE TESTBED

The proposed testbed consists of the implementation scheme for SCT in a networked hybrid system, comprised of multiple subsystems. The SCT is responsible for coordinating the subsystems, according to the specifications. The subsystems are distributed along a communication network, in which the events are transmitted.

The adopted implementation architecture is based on the one proposed initially by De Queiroz and Cury (2002). However, in their implementation, all the control logic is coded in a single device, a programmable logic controller (PLC). This is different from the proposed implementation scheme, shown in Fig. 4, where multiple devices, called *nodes*, are connected through a communication network.

The node global controller (equivalent to the controller of Fig. 2 and identified by the number (1) in the top right corner of the block global controller in Fig. 4) includes the modular supervisors (block (2)) and the global product system (block (3)), that is, it includes the models of all the subsystems. The supervisors obtained are local modular supervisors (De Queiroz and Cury, 2000) and, in order to reduce even more the size of the automata that have to be coded, a supervisor reduction technique is applied (Su and Wonham, 2004). Then, the information about which event is disabled or enabled in each of the states of the reduced supervisor is also obtained. All the preceding steps were done using the computational tool UltraDES (Alves et al., 2017). The modular supervisor (block (2)) is informed of the occurrence of all events in the system. Upon the reception of an event σ, each supervisor that has a transition labeled with σ will make the transition, updating their current state. Each state of the modular supervisors has associated a set of controllable events that are disabled. Note that self-loops can be ignored when coding the supervisors, since they do not change the current state and, consequently, the control action. If a particular event is disabled by at least one of the supervisors, then it is disabled and this information is sent to the product system level.

The product system (block (3)) controls the evolution of the system. The internal structure of this block includes all the models of the subsystems in addition to a component called event decision maker (EDM). The EDM receives from each subsystem model i the set $\Gamma_{ci}(q) = \Gamma_i(q) \cap \Sigma_{ci}$, corresponding to the set of feasible controllable events at state q of subsystem i. The set $\Gamma_c = \bigcup_{\forall i} \Gamma_{ci}(q)$ is the set of feasible controllable events of the overall system at a given moment. Note that this set may change whenever one of the subsystems changes its corresponding state. The EDM also receives the set of disabled events $\mathcal{D}$, information that is provided by the block modular supervisors. Whenever the set $\Gamma_c \backslash \mathcal{D}$ is not empty, the EDM will choose an event to be triggered. If the set $\Gamma_c \setminus \mathcal{D}$ is empty, the EDM waits for the reception of an uncontrollable event, which comes from the subsystems through the network. Upon reception of an uncontrollable event, all subsystem models and supervisors are updated. Note that this may change the sets $\mathcal{D}$ and Γ_c. How the choice is made by the EDM will depend on the implemented rule. We enumerate a few possibilities:

- Event priority: the EDM has information about which event has priority over the others;
- Predefined sequence: the EDM has an ordered list of events and the decisions are made according to this list. This makes sense when an optimization is in place and a best sequence is known (Pena et al., 2022; Alves et al., 2021);
- Random: the EDM chooses an event randomly.

Once a controllable event is chosen by the EDM, it will be sent into the network through the network interface (NI), represented by block (4) in Fig. 4, and sent back to the subsystem and supervisor models, allowing them to update their current state. All nodes connected to the network (blocks (5)) will receive the event, but if a node does not have that event defined in its alphabet, the event will be simply ignored. This selection of events is done by the network interface (blocks (6)). If the event is accepted into a node, it will be treated by the local product system (blocks (7)). The local product system is responsible for translating controllable events into commands and responses into uncontrollable events.

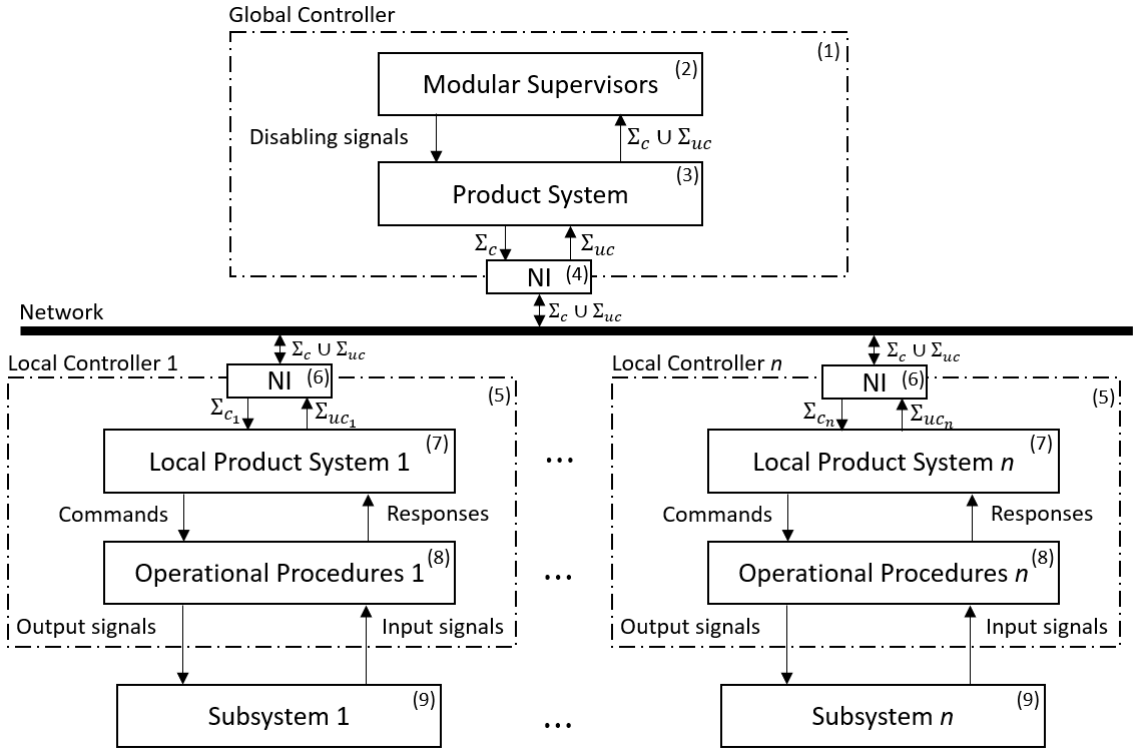


Fig. 4. Proposed implementation scheme.

Finally, the operational procedures level (blocks (8)) is responsible for mapping commands to the physical signals and the signals back to responses. A human operator can also interact with the system if we model the human-machine interface as a node that is also connected to the network. Events generated by such a node are uncontrollable from the global controller's point of view, since they cannot be prevented from happening.

The proposed implementation scheme is applicable to a wide range of industrial processes that have a centralized controller. Furthermore, the technology employed to implement it can vary from PLCs to micro-controlled based devices. Furthermore, the communication protocol does not necessarily need to be the same for all devices. The condition is that the controller has to be able to communicate with all devices. However, if different protocols are used, then the IDS has to be a much more complex structure.

5.1 Hardware

One of the features we wanted this implementation to have was low cost. To accomplish this, we employed arduino boards. Arduino is an open-source electronic prototyping platform and has multiple types of micro-controlled boards, with different number of input/output pins, flash and EEPROM memory and buit-in communication capabilities, among other specifications. One of the advantages of using arduino boards is the huge number of off-the-shelf modules that can be connected to them to add more functionalities.

Each one of the nodes in the system is based on an arduino board. In order to allow the communication between the boards we employed the Controller Area Network (CAN) protocol. It is a multi-master serial communication bus, and it is a network of independent controllers. For effective transmission, it follows reliable error-detection methods and, for arbitration on message priority and collision detection, it uses carrier sense multiple access protocol. Due to these reliable data transfer characteristics, this protocol has been in use in buses, cars and other automobile systems, factory and industrial automation, mining applications and others (Corrigan, 2016).

Although we chose to use the CAN protocol for the communication between devices, other protocols could be employed as well, such as Foundation Field Bus and Industrial Ethernet. One of the reasons for using CAN protocol was the easy access to low-cost arduino-compatible CAN modules, which allowed the devices to exchange information.

As the system was simulated in hardware, actuators like valves and the mixer are represented with LED's. The sensor level is implemented as a trimpot, which gives in its output a voltage in the range from 0 to 5V. The temperature of the liquid in the tank is represented as the output of a RC circuit while the control action related to the heater is applied in its input. Because the focus of this work was the implementation of a DES control, many simplifications were made related to the continuous-time variables. However, the implementation can be improved without requiring big changes in the DES control. The schematics of each node are available in an online repository [1] .

[1] https://lacsed.github.io/security-testbed/

The arduino boards are programmed using the arduino language, which is a "dialect" of C/C++. The core of the software is the SCT Library, presented next.

5.2 SCT Library

The SCT library, which was developed in the context of this work, provides a way to implement the control system using the automata as they are obtained. The main data type of this library is called DES and has two modes of operation, depending on the type of the node (global controller or local controller) it is running. The SCT library includes automata of the plants, with disjoint alphabets, and automata of the supervisors, with associated disablement information. The disablement information consists of the set of disabled events in each one of the states of the supervisor. If a controllable event is not disabled, then it is assumed to be enabled.

Upon initialization, the global controller will trigger a controllable event if there is at least one enabled or it will wait for the reception of an uncontrollable event. Once a controllable event is chosen, the event is sent into the network, the models are updated and the process repeats until there are no controllable events enabled. When an uncontrollable event is received, then the models are updated and this may result in a new set of enabled controllable events and the process repeats.

Regarding the local controller, when a controllable event is received, the models are updated and digital outputs can be set or reset. In the case of a continuous-time control or more complex sequences of discrete-event activities to be executed as an operational procedure, a flag can be set by the reception of an event and the underlying control is executed until the flag is reset. This can happen with the arrival of another controllable event or after a response is generated by the underlying controller or by the physical system.

The reception of events by the global controller or local controllers triggers an interruption, which in turn, calls a function to handle the incoming information. Initially, the controllability of the received event is verified (the global and local controllers only expect the reception of uncontrollable and controllable events, respectively) and then the models are updated if the event is defined for any of the automata in the node. Otherwise, the event is simply ignored. The complete source-code for all the nodes are available at the online repository.

Regarding the deception attacks, the SCT library offers access to the interface network of the nodes, represented by blocks (4) and (6) in Fig. 4. Thus, one can modify the behavior of the network interface in order to implement a deception attack or to implement a defense technique.

6. DISCUSSION

When implementing SCT over a system, several problems arise and have to be dealt with. In fact, Fabian and Hellgren (1998) and Dietrich et al. (2002), among others, identify the following problems:

(1) Causality: who generates which kind of event;
(2) Signals and events: is related to the translation and time behavior of the PLC variables and events as in SCT;
(3) The avalanche effect: corresponds to a multiple-state transition of a supervisor due to a single occurrence of an event;
(4) Inexact synchronization: is associated with the situation where the occurrence of an uncontrollable event that is observed by a supervisor with some delay invalidates the control action already established by such a supervisor;
(5) Choice: depending on the choice made in addressing a given event in PLC code, the actual behavior obtained under supervision may be blocking even when the supervisor is nonblocking.

Our proposed implementation deals with all the problems enumerated. The problem of causality is solved by assigning the generation of controllable and uncontrollable events to the global controller and local controllers, respectively. The translation of signals to events and events to signals is handled in the local controllers. The issue of having multiple events happening simultaneously is solved by the network's constraint of transmitting only one event at a time. The avalanche effect is not an issue in this implementation since the models are updated only once for each event.

The problem of choice is dealt within the product system level in the global controller. The EDM is responsible for choosing an event among a set of enabled events. The fact that a system under control may be blocking even when the supervisors are nonblocking is a modeling issue. Basically, to avoid the problem we have to guarantee that if a marked state is reachable by a sequence of uncontrollable events, then the system has to reach a state after the execution of controllable events where it can only wait for the occurrence of the uncontrollable ones. Otherwise, a marked state may never be reached if a sequence of uncontrollable events that lead the system to the marked state can be preempted at some point by the occurrence of a controllable event.

Our proposed solution does not suffer from the problem of inexact synchronization between plant and supervisor, which can occur when the communication delay is considerable when compared with the time constants of the system. This is not the case for our proposed implementation since events are treated as soon as they are received by the nodes and sent within a time window of few milliseconds, which is adjustable by the user. However, if delays are the subject of study, they can be forced to occur by changing the function that handles the reception and transmission of events.

7. CONCLUSION

Aiming to provide a testbed for security of DES related techniques, we present an implementation scheme based on low-cost devices that can be adapted easily enough to fit a wide range of control systems. The implementation offers access to the incoming and outgoing communication of its devices. This allows one to simulate an attack by deliberately changing the received and transmitted events and analyzing the impact of the attack on the system. For

future work, we intend to propose an architecture with decentralized controllers. Additionally, we plan to make a computational tool to automate the generation of code directly from the DES models.

REFERENCES

Alves, L.V.R., Martins, L.R.R., and Pena, P.N. (2017). Ultrades-a library for modeling, analysis and control of discrete event systems. *Proceedings of the 20th World Congress of the International Federation of Automatic Control*, 5831–5836.

Alves, L.V., Pena, P.N., and Takahashi, R.H. (2021). Planning on Discrete Event Systems Using Parallelism Maximization. *Control Engineering Practice*, 112(August 2020), 104813.

Corrigan, S. (2016). Introduction to the Controller Area Network (CAN). Technical Report. URL `www.ti.com/lit/an/sloa101b/sloa101b.pdf`. Last accessed: March 7th, 2022.

De Queiroz, M.H. and Cury, J.E.R. (2002). Synthesis and Implementation of Local Modular Supervisory Control for a Manufacturing Cell. *Proceedings of the 6th International Workshop on Discrete Event Systems*, 377–382.

De Queiroz, M.H. and Cury, J.E.R. (2000). Modular control of composed systems. In *Proceedings of the American Control Conference*, volume 6, 4051–4055.

Dietrich, P., Malik, R., Wonham, W.M., and Brandin, B.A. (2002). Implementation Considerations in Supervisory Control. In B. Caillaud, P. Darondeau, L. Lavagno, and X. Xie (eds.), *Synthesis and Control of Discrete Event Systems*, 185–201. Kluwer Academic Publishers.

Fabian, M. and Hellgren, A. (1998). PLC-Based Implementation of Supervisory Control for Discrete Event Systems. *Proceedings of the IEEE Conference on Decision and Control*, 3(December), 3305–3310.

Mourtzis, D., Angelopoulos, K., and Zogopoulos, V. (2019). Mapping Vulnerabilities in the Industrial Internet of Things Landscape. *Procedia CIRP*, 265–270.

Pan, X., Wang, Z., and Sun, Y. (2020). Review of PLC Security Issues in Industrial Control System. *Journal of Cyber Security*, 2(2), 69–83.

Pena, P.N., Vilela, J.N., Alves, M.R.C., and Rafael, G.C. (2022). Abstraction of the Supervisory Control Solution to Deal with Planning Problems in Manufacturing Systems. *IEEE Transactions on Automatic Control*, 67(1), 344–350.

Rashidinejad, A., Lin, L., Wetzels, B., Zhu, Y., Reniers, M., and Su, R. (2019). Supervisory Control of Discrete-Event Systems Under Attacks: An Overview and Outlook. In *Proceedings of the 18th European Control Conference*, 1732–1739. EUCA.

Su, R. and Wonham, W.M. (2004). Supervisor Reduction for Discrete-Event Systems. *Discrete Event Dynamic Systems*, 14, 31–53.

Yuan, Y., Yang, H., Guo, L., and Sun, F. (2019). *Analysis and Design of Networked Control Systems under Attacks*, volume 1. CRC Press, New York, 1st edition.