2025 IEEE 19th International Conference on Control & Automation (ICCA)
June 30 - July 3, 2025. Tallinn, Estonia

Detecting and Resolving Feature Interactions in Cyber-Physical Systems Using Formal Methods

H. D. Walker, S. L. Ricker and H. Marchand

***Abstract*— We investigate the use of formal methods to detect and resolve *feature interactions* (FI) in *cyber-physical systems* (CPS). These systems are often made up of multiple components that may interact with each other in unexpected and unwanted ways, potentially creating a security risk. Specifically, we use supervisory control theory to examine FI in a smart home, an example of a CPS. With the rising adoption of smart home devices, mitigating these interaction threats at the modelling stage is important before they are installed in homes. We present an extended taxonomy of FI threats that affect such a CPS. We demonstrate how one such threat, *chaotic device management* (*Codema*), can be detected and resolved in a system comprised of a smart light bulb that supports two disjointed device management channels.**

I. Introduction

The concept of *feature interactions* (FI) originated in software engineering. An FI occurs when two features are integrated, and the intended functionality is modified or negatively affects the overall system behavior. While FI research initially took place in the context of telephony systems [1], after nearly thirty years of research, the results are primarily limited to the specifics of those systems [2]. Recently, however, FI studies have moved to the domain of *cyber-physical systems* (CPS) [3].

CPS are systems that have both computational abilities and the ability to interact with their environment [4] and are part of the *Internet of Things* (IoT). CPS are far more complex than telephone systems, and are often made up of multiple components that perform different functions, and which may come from different manufacturers. As a result, these systems are vulnerable to FIs, and due to their complexity, much of the work done in telephony does not carry over to CPS. In this paper, we are interested in examining FIs in a specific instance of a CPS: a smart home IoT augmented with sensors and actuators.

Smart homes can include any number of Internet-enabled smart devices, which can interact with each other and their environment. The benefits of these interactions include convenience and improved energy efficiency [5]. However, with the increasing prevalence of smart homes, it is more important than ever that this technology is secure.

H. D. Walker and S. L. Ricker are with the Department of Mathematics and Computer Science, Mount Allison University, Sackville, NB, CANADA. Email: {hdwalker,lricker}@mta.ca

H. Marchand is with the University of Rennes, Inria, CNRS and IRISA, Rennes, FRANCE. Email: herve.marchand@inria.fr

We acknowledge the support of the Natural Sciences and Engineering Research Council of Canada (NSERC), [funding reference number 595677].

Cette recherche a été financée par le Conseil de recherches en sciences naturelles et en génie du Canada (CRSNG), [numéro de référence 595677].

FIs in smart home settings bear severe digital and physical security implications. The functionality of a smart home device is managed by a *device management channel* (DMC). Examples of DMCs include a device's proprietary app, or third-party management channels such as Amazon Alexa, Google Home, Apple Home, Zigbee, Z-Wave, and others. Many smart devices support multiple DMCs so that users can choose which one(s) they wish to use [6], [7]. DMCs that operate entirely independently of one another are said to be *disjointed*, and can leave systems vulnerable to FIs. In [6], this phenomenon is called *chaotic device management* (*Codema*). Notably, the authors were able to exploit disjointed DMCs in various smart devices, including smart locks and garage door openers. In a real-world setting this scenario could be catastrophic, potentially allowing bad actors to gain physical access to the user's home. In addition to degrading systems' security, FIs cause unwanted behaviors that violate consumers' expectations of how their devices should work and diminish their satisfaction [8].

Strategies already exist for detecting FIs in CPS: examples include mental models [9], model checking [10], and software solutions [11]. Formal methods have been used to model IoT [12], however, to our knowledge, they have not yet been used to examine FIs in this domain. We want to use formal methods to verify that a system is functioning correctly at the modeling stage. Such approaches reduce the cost of developing software [13] and make it possible to eradicate many kinds of bugs that lead to unsafe software, such as buffer overruns and flaws in protocols [14], [15]. Despite their potential positive impact, formal methods are underused in software development, in part due to their perceived difficulty [16].

To detect and resolve FIs in CPS, we propose applying *supervisory control theory* [17], where systems are typically modeled as finite automata. To facilitate modeling the complex behavior of CPS, we instead use *extended finite automata* (EFAs) [18]. Our aim is to address FI issues before the system is built, thus improving the security and reliability of these systems and the efficiency of their development.

In Section II, we present relevant background in EFAs, control theory, and modeling CPS. A taxonomy of FI threats that affect CPS, developed by [11], is discussed in Section III and extended in III-D. Section IV features an example of a control-theoretic approach to detecting and resolving an FI in a modeled CPS. Finally, in Section V we discuss the example as well as future research in using formal methods to study CPS.

II. Background and Notation

In this section, we present the notation and theoretical background necessary to discuss the taxonomy and our example.

A. Extended Finite Automata

In order to rigorously study the behavior of a complex system made up of discrete components, we use *extended finite automata* (EFAs) [18]. An EFA is a tuple

$$G = (Q, V, \Sigma, \Delta, q_0, Q_m),$$

where

- Q is a finite set of states;
- $V = \{v_1, \ldots, v_p\}$ is a set of p one-dimensional data variables, where each $v_i \in V$ is defined over domain D_i, for $i \in \{1, \ldots, p\}$;
- Σ is a finite set of events;
- $\Delta \subseteq Q \times Q \times \Sigma^* \times \mathcal{G} \times \mathcal{F}_{D \to D}$ is the transition relation (defined below), closed under concatenation, where $\mathcal{G}$ denotes the set of all boolean formulas over V, $\mathcal{F}_{D \to D}$ is the set of all functions from D to D, where $D = D_1 \times \ldots \times D_p$;
- $q_0 \in Q$ is the initial state; and,
- $Q_m \subseteq Q$ is the set of marked states.

Each transition $\delta \in \Delta$ is a tuple $\delta = (q, q', e, g, f)$, where

- $q, q' \in Q$ are the transition's source and destination states, respectively;
- $e \in \Sigma$ is the event that triggers the transition;
- $g \in \mathcal{G}$ is the enabling guard for the transition; and,
- $f \in \mathcal{F}_{D \to D}$ is the transition's data update function.

A transition $\delta = (q, q', e, g, f)$ will be carried out when e occurs if G is in state q and the guard g is `true`. After δ occurs, G will be in state q' and the data variables will be updated according to f.

We will find it useful to compose EFAs. The *parallel composition* of two EFAs G_1 = (Q_1,V_1,Σ_1,Δ_1,$q_{0,1}$,$Q_{m,1}$) and G_2 = (Q_2,V_2,Σ_2, $\Delta_2, q_{0,2}, Q_{m,2}$) is the EFA $G = (Q, V, \Sigma, \Delta, q_0, Q_m)$, where $Q = Q_1 \times Q_2$, $V = V_1 \cup V_2$, $\Sigma = \Sigma_1 \cup \Sigma_2$, $q_0 = (q_{0,1}, q_{0,2})$, and $Q_m = Q_{m,1} \times Q_{m,2}$, and Δ is defined as follows: $\forall (q_1, q_1', e, g_1, f_1) \in \Delta_1$, $\forall (q_2, q_2', e, g_2, f_2) \in \Delta_2$,

- $\forall e \in \Sigma_1 \setminus \Sigma_2$, $((q_1, q_2), (q_1', q_2), e, g_1, f_1) \in \Delta$;
- $\forall e \in \Sigma_2 \setminus \Sigma_1$, $((q_1, q_2), (q_1, q_2'), e, g_2, f_2) \in \Delta$; and,
- $\forall e \in \Sigma_2 \cap \Sigma_1$, $((q_1, q_2), (q_1', q_2'), e, g_1 \wedge g_2 \wedge [f_1|_{D_s} = f_2|_{D_s}], f_1 \oplus f_2) \in \Delta$,

where $f_i|_{D_s}$, for $i \in \{1, 2\}$, denotes the restriction of f_i to the domain D_s , which is the domain of the shared variables, *i.e.*, $V_1 \cap V_2$, and $\oplus$ denotes function composition.

B. Supervisory Control of EFAs

The goal of *supervisory control theory* [17] is to take a model of an uncontrolled system G and conform it to a specification of desired behavior G_s by synthesizing a controller, if one exists, that disables undesirable behavior that takes the system out of the specification into a state $q \in Q_f$, where $Q_f \subset Q$ is a set of forbidden states. Without loss of generality, we assume G_s is a subautomaton of G, with $G_s = (Q_s, V_s \subseteq V, \Sigma, \Delta_s \subseteq \Delta, q_0, Q_{m,s} \subseteq Q_m)$, where $Q_s = Q \setminus Q_f$. We employ the algorithm provided in [18] to modify G in place, updating guards on transitions in Δ that may bring the system into a forbidden state in Q_f. A sketch of this algorithm is shown in Algorithm 1.

The existence of a controller is dependent on the *controllability* of G_s. To facilitate the control of G, we partition Δ into the set of controllable transitions Δ_c and the set of uncontrollable transitions $\Delta_{uc} = \Delta \setminus \Delta_c$. The supervisor can only update the guards of transitions in Δ_c. Given an EFA G, a specification G_s, a set of forbidden states Q_f, and a set of controllable transitions Δ_c, we say a state $q \in Q$ is (adapted from [18]):

- *nonblocking* if it is possible to reach a state $q' \in Q_m$ from q via a sequence of transitions $\delta \in \Delta$, taking into consideration guards and updates to variables;
- *safe* if $q \in Q_s$; and,
- *controllable* if q is safe and there are no uncontrollable transitions $(q, q', e, g, f) \in \Delta_{uc}$ with $q' \in Q_f$

Then G_s is nonblocking, safe, and controllable if for every $q \in Q_s$, q is respectively nonblocking, safe, and controllable, again considering guards and updates to variables.

Algorithm 1 (From [18]) Sketch of algorithm for supervisor synthesis using EFAs.

Input: An EFA $G = (Q, V, \Sigma, \Delta, q_0, Q_m)$ with a set of forbidden states $Q_f \subset Q$ and a set of controllable transitions $\Delta_c \subseteq \Delta$.

1: Create and initialize the *nonblocking predicate* N_q and *bad location predicate* B_q to `false` for all $q \in Q$
2: For each $q \in Q$, change N_q to `true` if $q \in Q_m$ or if it is possible to reach a state q' from q so that $N_{q'} =$ `true`. Repeat this step until there are no changes to N_q for any $q \in Q$.
3: For each $q \in Q$, change B_q to `true` if $q \in Q_f$, $N_q =$ `false` or there is a transition $\delta \in \Delta_{uc}$ that could take the system from q to a state q' that has $B_{q'} =$ `true`. Repeat this step until there are no changes to B_q for any $q \in Q$.
4: For each transition $(q, q', e, g, f) \in \Delta_c$, change g to `false` if q' has $B_{q'} =$ `true`.
5: If there were changes to any guards, return to step 1. Otherwise stop.

Result: Controllable transitions $\delta \in \Delta_c$ that may result in the system entering a forbidden state have guard $g =$ `false` and are thus disabled.

C. modeling CPS

Now that we have discussed relevant background in EFAs and supervisory control, we now discuss our approach to modeling the behavior of CPS using EFAs. In [8], [11], the behavior of cyber-physical systems is modeled using *trigger-condition-action* (TCA) rules. A rule R is modeled as a tuple $R = (T, C, A)$, where T is the *trigger*, $C = C_1 \wedge C_2 \wedge \ldots \wedge C_k$

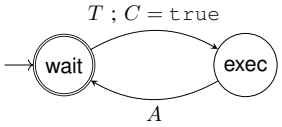

Fig. 1. A TCA rule modeled as an EFA.

is a set of boolean *conditions*, and A is an *action*. When event T occurs, if the condition C is satisfied, the action A will be performed. Unwanted interactions between TCA rules on one or more devices are called *feature interactions* (FIs) and are the basis for the taxonomy presented in Section III.

We model a TCA rule using an EFA of the form of Fig. 1 by letting $T, A \in \Sigma$ and $C \in \mathcal{G}$. Here, the EFA begins in the wait state and moves to the exec state when T occurs and C is $\texttt{true}$. Event A then occurs, bringing the EFA back to the wait state. As the EFA transitions back to the wait state, it may update data variables according the transition's data update function f_A. The ability to model the behavior of a CPS in terms of TCA rules using EFAs will allow us to identify feature interactions in modeled CPS and resolve them using supervisory control.

III. Taxonomy

Noting our approach to modeling CPS as EFAs, we now model different FI threats that affect CPS, in order to facilitate their detection and resolution. A *cross-app interference threat* occurs in a CPS when there are two TCA rules $R_1 = (T_1, C_1, A_1)$ and $R_2 = (T_2, C_2, A_2)$ that may or may not belong to different devices or pieces of software in the system, and A_1 interferes with T_2, C_2, or A_2. A taxonomy of these threats is provided in [11]. These threats are divided into three categories: *action-interference threats*, *trigger-interference threats*, and *condition-interference threats*. In this section, we present these categories as well as several sub-categories of threats, and extend the taxonomy with a new category of threat.

A. Action-Interference Threats

An action-interference threat occurs when two rules R_1 and R_2 act on the same actuator, and their actions A_1 and A_2 have contradictory effects on the environment when triggered simultaneously. As such, this category of threat is effectively a race condition, and is thus not amenable to being modeled using automata, as our EFA models do not take time into account. We will therefore not model or discuss this class of threat.

B. Trigger-Interference Threats

Trigger-interference threats occur when one rule triggers another. They are further divided into three subcategories:

Covert Rules When the result of a rule R_1's action triggers R_2, *i.e.* $A_1 = T_2$, and the conditions are such that R_2's action will execute after that of R_1, *i.e.* $C_2 = \texttt{true}$, this creates a *covert rule*, where R_1's trigger and conditions result in both R_1 and R_2's actions being carried out. An example of such an interaction would be R_1 causing a window to open when the CO_2 level in a room rises above a certain level, and R_2 causing the heat to turn on when the window is open. The result is a covert rule that the heat will come on when the CO_2 level is above a certain level. This example is modeled using EFAs in Fig. 2.

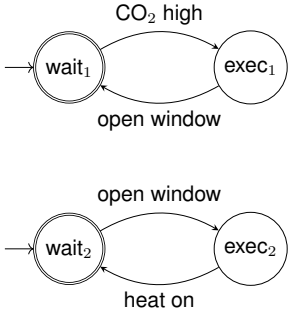

Fig. 2. Covert rule threat: CO_2 high covertly causes heat on to occur.

Self Disabling A self disabling threat is a variation of a covert rule where R_2's action disables R_1. In other words, $A_1 = T_2$ and $f_{A_2}(\{C_1 \in \{\texttt{true}, \texttt{false}\}\}) = \{C_1 = \texttt{false}\}$. For example, R_1 may turn on the air conditioner when the temperature rises above a set level, and a R_2 may cut power to the air conditioner when the monthly power consumption exceeds a certain threshold [11]. The result is that, under certain conditions, the temperature rising above the set level will ultimately result in power being cut to the air conditioner, disabling R_1. This example is modeled using EFAs in Fig. 3.

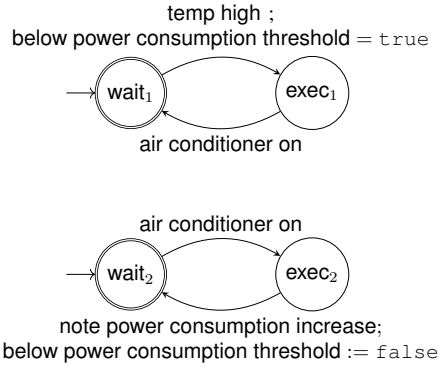

Fig. 3. Self-disabling threat.

Loop Triggering R_1 and R_2's conditions are both met, and their actions trigger each other, *i.e.* $A_1 = T_2$ and $A_2 = T_1$, and A_1 and A_2 contradict each other. A trivial example would be one rule that opens a window every time it is closed, and another rule that closes the same window every time it is opened. This threat is modeled using EFAs in Fig. 4.

C. Condition-Interference Threats

Condition-interference threats occur when one rule's action enables or disables other rules. They are further divided into two categories:

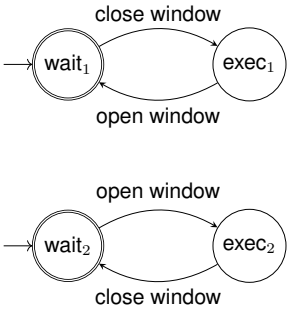

Fig. 4. Loop-triggering threat

Disabling-Condition R_1's action causes R_2's condition to cease to be satisfied, *i.e.* $f_{A_1}(\{C_2 \in \{\texttt{true}, \texttt{false}\}\}) = \{C_2 = \texttt{false}\}$. An example of a disabling condition threat would be R_1 turning on an exhaust fan when the stove temperature is above $200°C$, and R_2 sounding an alarm when smoke is detected and the exhaust fan is off. This threat is modeled using EFAs in Fig. 5.

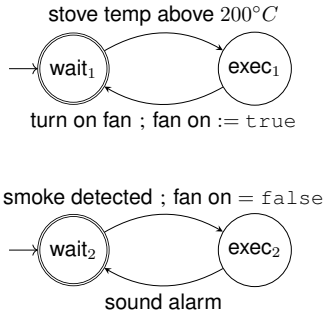

Fig. 5. Disabling-condition threat

Enabling-Condition R_1's action causes R_2's condition to become satisfied, *i.e.* $f_{A_1}(\{C_2 \in \{\texttt{true}, \texttt{false}\}\}) = \{C_2 = \texttt{true}\}$. An example of such a threat would be as follows: R_1 turns on an outdoor porch light when it is dark outside, and R_2 turns on a security camera when motion is detected and the porch light is on. This threat is modeled using EFAs in Fig. 6.

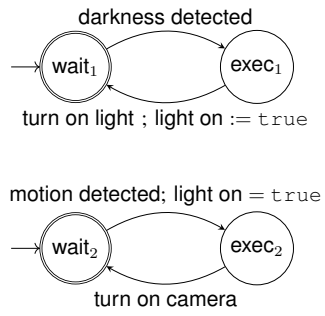

Fig. 6. Enabling-condition threat

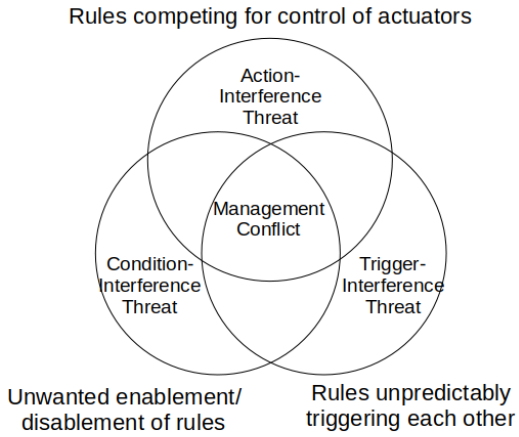

Fig. 7. Management conflicts display characteristics of all three threat categories.

D. Extension of the Taxonomy

We propose adding another type of cross-app interference threat to this taxonomy, caused by poorly-behaved DMCs:

Management Conflict The device owner/manager cannot control or tell which rules are enabled and disabled on devices in the system. An example of such a threat, called *chaotic device management (Codema)* is presented in [6] and was successfully exploited by the authors on multiple popular smart home devices. This threat occurs when a device supports multiple *device management channels* (DMCs), but the pieces of software running on the device to support the different DMCs are *disjointed*: there is no communication between the DMCs to establish which ones should be enabled and which should be disabled. The result is a device that can be covertly paired with and controlled by a malicious actor without the owner's knowledge.

This type of threat transcends the other three categories, as illustrated in Fig. 7; such a threat may result in bad actors competing for control of actuators (as in an action-interference threat), introducing new rules that may trigger or be triggered by others (as in a trigger-interference threat), or spontaneously enabling and disabling existing rules (as in a condition-interference threat).

In this section, we have introduced the taxonomy developed by [11], extended it to include a new class of threat, and included EFA models of each threat in the taxonomy. Having a taxonomy of threats and accompanying EFA models will make identifying these in modeled systems possible, allowing us to leverage supervisory control to disable behaviors that lead to these conflicts.

IV. Example

This example demonstrates an application of EFAs to the detection and resolution of a management conflict in a model of a simple CPS: a smart light bulb that supports two disjointed DMCs. Such devices are increasingly common, a

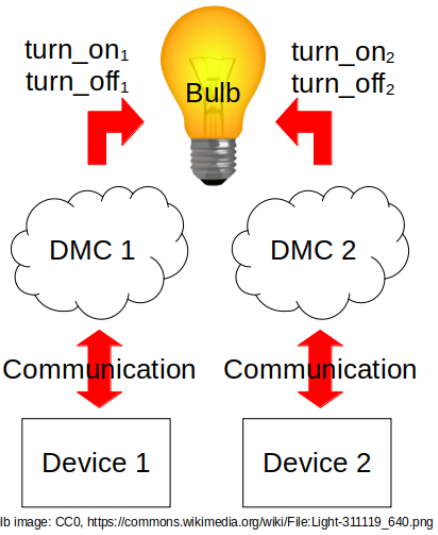

Fig. 8. Example of a smart light bulb that supports two disjointed DMCs.

notable example being Philips Hue products which support several different device management channels [19].

We model our bulb with the EFA in Fig. 9 and model each DMC with the EFA in Fig. 10, where i identifies each unique DMC ($i \in \{1, 2\}$). DMC i begins in the $\mathsf{unpaired}_i$ state with $\mathsf{enabled}_i = \texttt{true}$, and then moves into the $\mathsf{waiting}_i$ state after being paired with if $\mathsf{enabled}_i = \texttt{true}$. It will then wait for instructions from whatever device has paired with it. A $\mathsf{turn\_on}_i$ event will occur when it is instructed to turn the bulb on, and a $\mathsf{turn\_off}_i$ event will occur when it is instructed to turn the bulb off. The smart bulb itself begins in the off state, and then turns on or off when instructed to do so by either DMC. An informal diagram of this system to aid intuition is shown in Fig. 8.

The model of the entire system is obtained by taking the parallel composition of the EFAs for the bulb and the two DMCs and is shown in Fig. 11. For this example, we assume that all transitions are controllable and that all states are marked.

For this example, our specification is that behavior which results in more than one DMC being paired is illegal; if it is possible for an unknown third party to pair to the bulb and control it without the owner's knowledge after it has been paired to the owner's preferred DMC, then that constitutes a management conflict and is an example of *Codema*. Then the two states in Fig. 11 in which both DMCs are paired are forbidden (elements of Q_f), and we draw them in red. Our specification G_s is then the same automaton with these states removed. We seek to synthesize a supervisor to restrict the behavior of G to that allowed by G_s, and do so by employing Algorithm 1. The result is that transitions into the states where both DMCs have been paired are disabled, shown in Fig. 12.

V. Discussion

In the bulb example in Section IV, by modeling each component individually and then composing them to obtain

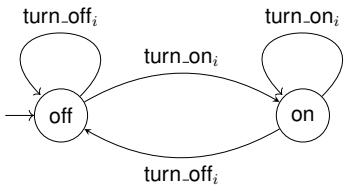

Fig. 9. Model of a smart light bulb.

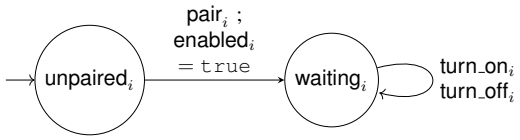

Fig. 10. Model of smart light bulb DMCs 1 and 2.

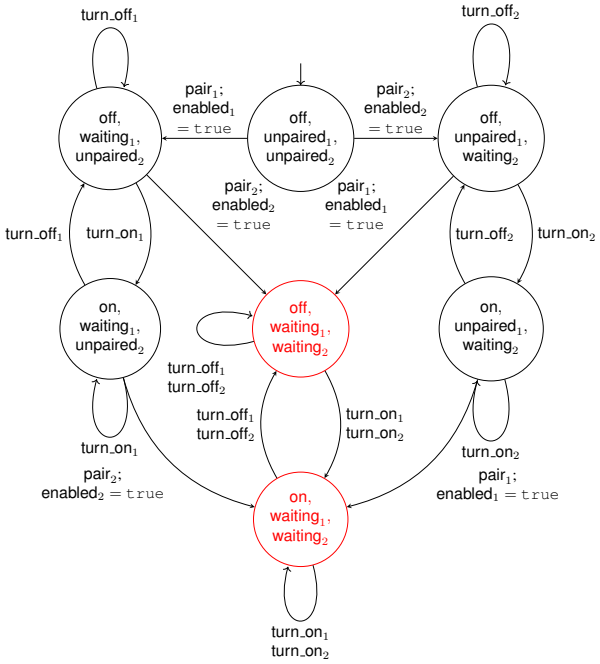

Fig. 11. Combined plant G and specification G_s. Red states are in Q_f.

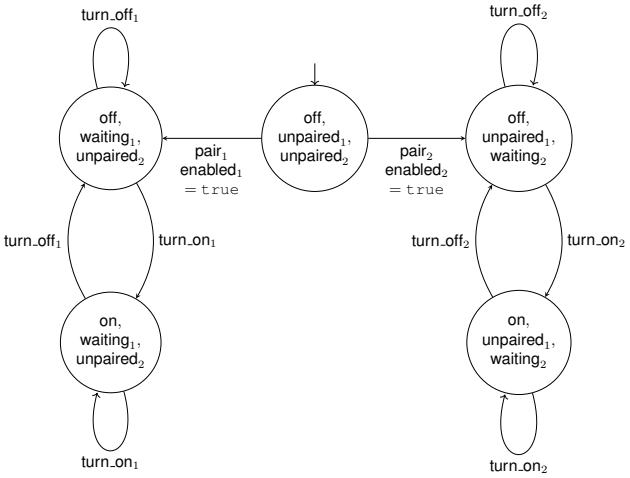

Fig. 12. System after carrying out Algorithm 1. Disabled transitions and now-inaccessible states are removed.

a model of the system, we were able to detect FIs and synthesize a supervisor to remove them, all before the system was implemented. In this example, we assume that all events are controllable. This is because, in this case, both DMCs and the mechanism for turning the bulb on and off would be pieces of software running on the smart bulb's computer, and their activity would be able to be monitored through its operating system. Such a supervisor could be implemented as a piece of software on the bulb given permission to disable the software for one of the DMCs, block communication with it, or otherwise prevent it from pairing after the other has been paired with, as in Algorithm 2. In some situations, however, it may not be the case that all of the undesirable behavior can be controlled by one device.

Algorithm 2 Example implementation of supervisor from Section IV.

Input: Event $e \in \Sigma$ in G from Fig. 11.

1: **if** $e = \mathsf{pair}_1$ **then**
2: $\quad \mathsf{enabled}_2 := \texttt{false}$
3: **else if** $e = \mathsf{pair}_2$ **then**
4: $\quad \mathsf{enabled}_1 := \texttt{false}$
5: **end if**

Result: It is not possible for both DMCs to be paired at the same time, and the management conflict is thus resolved.

In this paper, we used EFAs as our underlying model. One limitation of EFAs, as with most varieties of finite automata, is that they are static. This makes it challenging to model a system which devices may enter and leave at will, a common situation in smart homes and other IoT systems. *Dynamic input/output automaton* (DIOA) [20] support such dynamics. This kind of automaton has not yet been used to study CPS. It is also of interest to apply strategies for the control of reconfigurable software systems as introduced in [21] to IoT systems. Additionally, due to the lack of consideration of time in EFAs, we were not able to model action-interference threats, which are analogous to race conditions. It is of interest to us to find a model that will allow us to model this class of threat.

In Section III, we extended the taxonomy presented in [11], and plan to produce examples of using supervisory control to resolve FIs from the remaining categories. We also plan to study how real-life FIs in the literature fit into these categories and add more categories and examples as necessary.

In this paper, we briefly discussed the benefits of applying formal methods to the study of FIs in CPS and presented a taxonomy of cross-app interference threats that affect CPS. Our contributions are an extension to said taxonomy to include device management conflicts caused by disjointed DMCs, EFA models of the threats in the taxonomy, and an example of employing EFAs and supervisory control theory to detect and resolve one such conflict in a model of a realistic CPS. With these contributions, we hope to improve the overall security of CPS by addressing FIs at the modeling stage.

REFERENCES

[1] P. Zave, "Feature interactions and formal specifications in telecommunications," *IEEE Computer*, vol. 26, no. 8, pp. 20–30, 1993.

[2] S. Apel, J. Atlee, L. Baresi, and P. Zave, "Feature interactions: The next generation," Dagstuhl Reports, 2014.

[3] B. Gafford, T. Dürschmid, G. Moreno, and E. Kang, "Synthesis-based resolution of feature interactions in cyber-physical systems," in *5th IEEE/ACM International Conference on Automated Software Engineering*, 2020.

[4] R. Baheti and H. Gill, "Cyber-physical systems," *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[5] B. K. Sovacool and D. D. F. Del Rio, "Smart home technologies in Europe: A critical review of concepts, benefits, risks and policies," *Renewable and sustainable energy reviews*, vol. 120, p. 109663, 2020.

[6] Y. Jia, B. Yuan, L. Xing, D. Zhao, Y. Zhang, X. Wang, Y. Liu, K. Zheng, P. Crnjak, Y. Zhang *et al.*, "Who's in control? on security risks of disjointed IoT device management channels," in *Proceedings of the ACM SIGSAC Conference on Computer and Communications Security*, 2021, pp. 1289–1305.

[7] B. Hammi, S. Zeadally, R. Khatoun, and J. Nebhen, "Survey on smart homes: Vulnerabilities, risks, and countermeasures," *Computers & Security*, vol. 117, p. 102677, 2022.

[8] B. Huang, D. Chaki, A. Bouguettaya, and K.-Y. Lam, "A survey on conflict detection in IoT-based smart homes," *ACM Computing Surveys*, vol. 56, no. 5, pp. 1–40, 2023.

[9] S. Yarosh and P. Zave, "Locked or not? mental models of iot feature interaction," in *Proceedings of CHI*, 2017, pp. 2993–2997.

[10] R. Trimananda, S. A. H. Aqajari, J. Chuang, B. Demsky, G. H. Xu, and S. Lu, "Understanding and automatically detecting conflicting interactions between smart home IoT applications," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1215–1227.

[11] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *IEEE/IFIP International Conference on Dependable Systems and Networks (DSN)*. IEEE, 2020, pp. 411–423.

[12] M. Zhao, G. Privat, E. Rutten, and H. Alla, "Discrete control for the internet of things and smart environments," in *8th International Workshop on Feedback Computing (Feedback Computing 13)*, 2013.

[13] E. M. Clarke and J. M. Wing, "Formal methods: State of the art and future directions," *ACM Computing Surveys (CSUR)*, vol. 28, no. 4, pp. 626–643, 1996.

[14] E. Jaeger, "Study of the benefits of using deductive formal methods for secure developments," Ph.D. dissertation, Université Pierre et Marie Curie-Paris VI, 2010.

[15] M. Oliveira, L. Ribeiro, É. Cota, L. M. Duarte, I. Nunes, and F. Reis, "Use case analysis based on formal methods: an empirical study," in *Recent Trends in Algebraic Development Techniques: 22nd International Workshop, WADT 2014, Sinaia, Romania, September 4-7, 2014, Revised Selected Papers 22*. Springer, 2015, pp. 110–130.

[16] M. Gleirscher and D. Marmsoler, "Formal methods in dependable systems engineering: a survey of professionals from Europe and North America," *Empirical Software Engineering*, vol. 25, pp. 4473–4546, 2020.

[17] P. J. Ramadge and W. M. Wonham, "Supervisory control of a class of discrete event processes," *SIAM J. Control Optim.*, vol. 25, no. 1, pp. 206–230, 1987.

[18] L. Ouedraogo, R. Kumar, R. Malik, and K. Akesson, "Nonblocking and safe control of discrete-event systems modeled as extended finite automata," *IEEE Transactions on Automation Science and Engineering*, vol. 8, no. 3, pp. 560–569, 2011.

[19] Philips. Smart lighting. [Online]. Available: https://www.philips-hue.com/en-ca

[20] P. C. Attie and N. A. Lynch, "Dynamic input/output automata: A formal and compositional model for dynamic systems," *Information and Computation*, vol. 249, pp. 28–75, 2016.

[21] N. Berthier, F. Alvares, H. Marchand, G. Delaval, and E. Rutten, "Logico- numerical control for software components reconfiguration," in *IEEE Conference on Control Technology and Applications*, 2017, pp. 1599 – 1606.