Received: July 25, 2022. Revised: September 4, 2022. 350

International Journal of
Intelligent Engineering & Systems

http://www.inass.org/

Conflict Detection and Resolution Model for Multi-users IoT Automation Systems

Hamada Ibrhim[1] Hisham Hassan[2] Ahmad B. Alkhodre[3] Emad Nabil[3]*

[1]*Faculty of Computers and Information, Minia University, Minia, Egypt*
[2]*Faculty of Computers and Artificial Intelligence, Cairo University, Giza, Egypt*
[3]*Faculty of Computer and Information Systems, Islamic University of Madinah, Madinah, Saudi Arabia*
* Corresponding author's Email: emadnabil@iu.edu.sa

Abstract: Building automation (BA) strives to control interconnected physical devices by using software management systems on which end-users can personalize their environmental preferences. In large buildings, among the leading causes of IoT apps conflict are the shareable locations/devices amongst residents and the diversity of their preferences. Addressing such conflicts and ensuring the safety of residents are vital requirements in building automation systems. Consequently, the potential of IoT safety and correctness frameworks relies on supporting conflict detection. This paper provides a model for detecting and resolving IoT automation conflicts. These conflicts can result from shareable locations or devices. The proposed model is evaluated using a benchmark dataset and refined scenarios collected from competitor-related works (80+ IoT apps with 117+ rules). The proposed model surpasses state-of-the-art models by covering more conflicts (joint behavior conflict); moreover, it does not require events' chain between IoT apps like other models, one more advantage is that the proposed model uses a filtering process in conflict detection which leads to small detection run-time. Thereby, our proposed model can maximize the correctness and safety of building automation systems.

Keywords: End-user programming, Event-condition-action programming, IoT apps, Conflict resolution, Correctness and safety, Energy saving.

1. Introduction

Building automation system (BAS) plays an important role and gains more attention in industrial and personal uses [1-3]. Optimal and safe benefits of these systems require a high level of awareness of its integrated resources (i.e., sensors, actuators, or any computing resource) that gain an artificial sensory perception of these buildings and their occupants' behavior s that convert the building into an intelligent ecosystem [4].

BAS provides several features: controlling lights, climate, HVAC, safety and security, comfort, energy-saving, and entertainment. Mainly, BAS is capable of orchestrating the use of these features through the use of customizable logic [5, 6]. This customizable logic is typically represented as if-this-then-that (IFTTT), or event-condition-action (ECA) rules [7-9], and it corresponds to users' behavior s (users' contexts). These rules are responsible for the influx of events and the contextual information of devices and environment collaborations [10, 11].

The existence of multiple contexts of different users in BAS has a high probability of occurrence [12, 13]. Multiple contexts happen due to the overlapping in time-horizon, shared location between users, and differences in their preferences [14]. When these users' IoT automation apps are activated at the same time, their effects on the indoor environment create a joint behavior situation. Occurring such situations may violate the environment requirements (a.k.a., invariants, policies, or constraints) [15, 16]. With the increasing number of users and their intentions' complexity, it is essential to ensure concurrency between different users do not result in conflicts. These users' services concurrency exacerbates dangerous behavior s or privacy risks [14, 17-20]

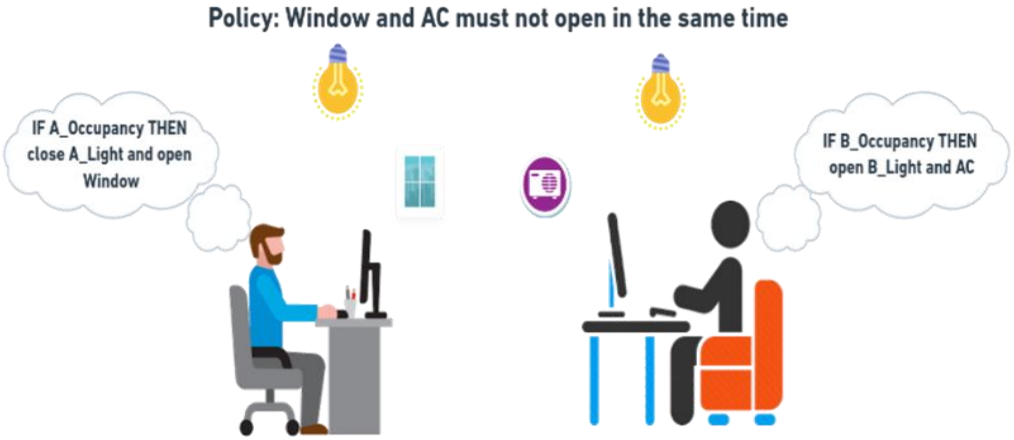

Figure. 1 A joint-behavior situation conflict for smart office

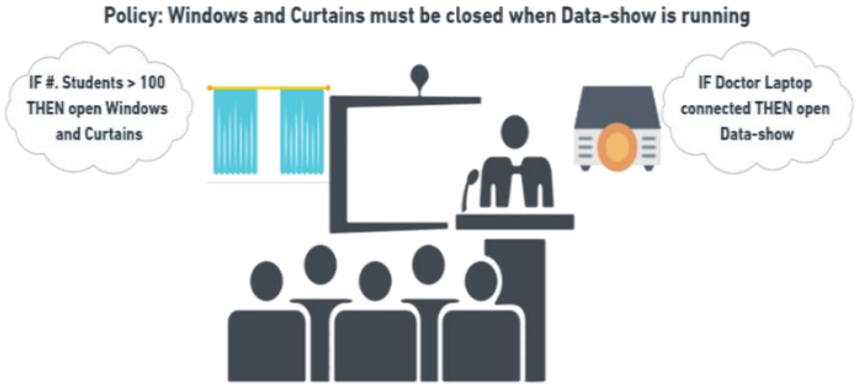

Figure. 2 A joint-behavior situation conflict for smart lecture hall

such as closing the main door while a fire ignited and frequently turning on and off AC. Ideally, these amiss interactions and users' services concurrency considered a challenge for the user [21] should be detected and resolved.

This paper proposes a prototype implementation for detecting joint behavior conflicts against the environmental requirements. Our proposed prototype uses satisfiability modulo theories (SMT) model checking [22] to analyze the joint behavior situations regardless of the number of IoT automation apps and policies involved. The proposed algorithm grants less detection time due to the filtering process used in conflict detection.

The remainder of the paper is organized as follows. In section 2 some motivation scenarios have been described that show the main focuses of this paper. After that, section 3 presents some related work on IoT inter apps conflict detection frameworks. The proposed conflict detection and resolution are explained in section4. To demonstrate the applicability of the proposed prototype, experimental results are presented in section 5. Finally, section 6 concludes the paper and describes future work.

2. Motivation scenarios

In this section, we discuss two canonical scenarios that highlight the focus of this paper. These scenarios may occur in different buildings that utilize system policies to control and define the sets of admissible and inadmissible situations in their automation.

In scenario 1 as shown in Fig. 1, suppose there are two users in a shareable workplace (e.g., smart office). The first user, A, has a personalized IoT automation app to open his light and allow fresh air by opening the window when A detected in this workplace. The second user, B, has a different behavior to opening his light and improving the indoor environment quality by opening AC when B is detected in this workplace. But there is a coexist situation that could occur simultaneously or at the overlapping time (i.e., A's IoT app begins at 10:00 AM for 2 hours, and B's IoT app is activated at 11:00 for 1 hour). In this situation, the overall system context can be described

as a joint of multiple effects of different IoT apps, as lights, AC, and windows opened simultaneously according to this scenario. If we supposed that there is an enforced safety environment policy that ensures the performance of AC in the workplace stated that "AC and window must not be opened at the same time." So, this coexists situation will cause a policy violation conflict.

In scenario 2 as shown in Fig. 2, suppose a lecture hall is customized by two users. The first user is the lecture hall preservative, who needs to manage the lecture hall temperature quality when the students' count is greater than 150 by opening the windows and curtains. The other user is the lecturer, who creates an IoT app to automatically open the data shown when his laptop is plugged in. The co-existence situation for both users may be happened due to overlapping time. If we supposed that there is an environment setting policy controls devices in the lecture hall when a lecture is beginning stating that "Windows and curtains must be closed when data-show is running." So, this coexists situation between lecturer and lecture hall preservative will cause policy violation conflict.

Although these scenarios are simple, it effectively illustrates the conflicts that could result when the IoT automation system has a multiple-users joint situation. Accordingly, scenarios with similar environmental influences could happen in smart buildings in which devices, locations, and time points are shared between different users. This conflict is mentioned by previous work [23, 24, 25] as implicit interference, the opposite- environment conflict and safety property, respectively. An environment property (e.g., temperature) is affected in opposite directions, for instance, when both AC and heater are open simultaneously. Unfortunately, these works require acquiring knowledge of devices, the service usage requirements history, or specifying rigidity policies. These requirements may make the detection of these conflicts more complex and consume time overhead.

3. Related work

This section investigated a set of related work for IoT apps automation conflicts detection and resolution either in single-user IoT apps or multiple users' IoT apps interactions.

Palekar et al. [26] explained with an empirical study some recommendations for trigger-action programming interfaces to detect and resolve user errors. They categorized users' errors into nine classes. Unfortunately, they did not consider the error that results from violating system policy by composite services of different users.

Shah et al. [27] provided a conflict detection schema for detecting and resolving rule conflicts (e.g., execution, shadow, and independent conflicts) and incompleteness. The work presented the concept of "anti-rule" which describes the opposite of a rule. However, the approach did not care about violating system invariants that govern the environment.

IoT Composer tool proposed in [28], which supported different IoT application development steps such as design, composition, verification, and deployment. The verification step checked the compatibility and absence of deadlocks to ensure bug-free compositions of objects. However, the tool did not provide how to solve these problems, has a degree of complexity for end-users to use, and did not concern with system policies.

In [29], an iRULE tool is provided to detect the inter-rule vulnerabilities that represent security risks in IoT apps that build based on the trigger-action rule style. Also, it provided a method based on natural language processing (NLP) to infer information flows. Although they provided a set of inter-rule vulnerabilities, there is no solution for these vulnerabilities. In addition, using NLP may imply overhead in time for complex IoT apps.

OKAPI platform is proposed in [30] to avoid conflicts that may be occurred when accessing or modifying shared resources in intelligent homes. These conflicts like consistency deficiencies, event reordering, and race conditions. The main drawbacks of OKAPI are that it did not have a representation of system requirements and its violation checks. Lee and Lin [31] discussed the multi-user activities conflict in a smart home. The situation awareness of users' activities is detected using wearable devices. The work proposed a conflict resolution algorithm that is based on some variables such as identity and time. The work is missing a factor affecting the interaction between multi-users, which is the environment constraints. Also, the variables used to resolve conflicts are the tool's responsibilities, limiting the user role.

In [23], investigated the difference between explicit and implicit interference problems between rules in a smart home. Explicit interference refers to the contradiction of conflict between multiple rules over a single actuator. On the other hand, implicit interference refers to the contradictory environmental effects caused by various rules over various actuators. A3ID (automatic interpretable implicit interference detection) method is proposed to detect these problems based on knowledge graphs and NLP. However, the A3ID did not provide resolution for the interference problems detected.

Functional and non-functional conflicts are investigated in [24] using a framework for detecting them in IoT services in shared locations between multiple users. The work provided conflict ontology and ontology rules that describe how the conflict occurs. However, the conflict detection based on user history of service usage frequency makes the work unsuitable for the real-time behavior of IoT automation services, where users create or customize services that need to check in real-time.

The "wireless context" concept is proposed in [32] which corresponds to the packets' workflow in IoT apps. Based on a machine learning model, the user IoT context and the wireless context are compared to detect anomalies such as App misbehavior, event spoofing, over-privilege, device failure, and hidden vulnerabilities. No solutions are provided for the vulnerabilities between IoT app interactions, and no representation of system environmental constraints.

In [33], a service composition model that is restricted by policy ontology is presented. The composite service is created for only one user and contains multiple web services over physical devices. Although the tool has the meaning of policy to adjust the system behavior dynamically, there is no check against policies for the generated composite service. Han et al. [34] proposed a policy-based approach for dynamically composing services based on context-awareness. The hybrid service corresponds to a user preference. No check if the generated composite service causes any conflicts or not.

In [35], investigated the interactions between policies (i.e., system axiom policy and dynamic behavior policy) in smart homes through the use of IRIS (identifying requirements interactions using semi-formal methods) as a feature interaction model. Conflict detection is based on converting system requirements (policies) to graphical notations as a set of tables and graphs and analyzing them according to an interaction taxonomy of guidelines. However, conflict detection (interaction detection) is performed using human developers. This requires overhead in time and effort to detect interactions in complex and heterogeneous IoT systems with hundreds of thousands of policies.

Nguyen et al. [36] proposed IoT sanitizer (IoTSAN) framework for detecting unsafe interaction events resulting from violating user-defined safety properties using the SPIN model checker [37]. However, the framework requires IoT program analysis and modifying overheads. Also, to build the dependency graph using IoTSAN, the IoT apps must have an event with contradicting values (i.e., On/Off events), and it fails to detect conflict in IoT apps (e.g., scenarios in section' motivation') not satisfying this condition.

Using an abstraction module, the RemedIoT framework proposed in [38] to detect and resolve IoT app conflicts concerning policies. Racing events and cyclic events are the main conflicts detected and resolved by RemedIoT. Although RemedIoT has the capability for remedial actions, it fails to detect conflicts in event services that depend on integer devices (e.g., "IF home mode THEN acThermostat = 18" and "if home temperature > 30 then acthermostate = 20"), since it supports only event services with ON or OFF states. Also, RemedIoT did not support combined event services conflict with policies.

Soteria [39] and IoTGuard [16] are two frameworks for IoT app verification. In Soteria, static analysis is performed to detect violations against identified properties. On the other hand, IoTGuard, safety, and security proprieties are checked during run-time IoT app interactions. Both frameworks are based on model checking (e.g., NuSMV [40]). The unified dynamic model defined in IoTGuard requires the IoT apps to have a shared events chain between them which consider a limitation in IoTGuard.

The security and safety of cross-app IoT interaction policies are investigated in [41] using the process calculus. The interaction policies defined in their work allow the detection of syntactic and semantic conditions for IoT apps, but they did not consider environmental requirements when two systems of apps interact.

3.1 Conventional conflict checking techniques weaknesses

To the best of our knowledge, the main weaknesses of the conventional conflict checking techniques are:

1. Some works are suitable for a limited number of IoT apps, which restricts their techniques' scalability.
2. Detecting inter-IoT apps interference is limited to specific device types, which limits device heterogeneity and coverage of their techniques.
3. Some works oblige the IoT apps to form an event chain between them to detect the inter-IoT apps conflict, which constraints their frameworks to detect other conflicts under different circumstances.
4. Ignoring the system policies for ensuring system correctness, adds a limitation to their frameworks, where these policies consider an

```
{
  "ServiceID": 1,
  "UserID": 123,
  "UserPriority": 1,
  "StartDate": "2022-1-1",
  "Time": "10-00-00",
  "Period": 120,
  "RepeatEvery": ["Su", "Tu"],
  "EndDate": "2022-3-18",
  "Rules":[
          {
           "Ser_ID": 1,
           "R_ID": 1,
           "Priority": 1,
           "RLoc": "Office5",
           "conditionGroup":[{"DeviceName": "Office5_BAoccupancy",
                  "value": "True"}],
           "actionGroup":[{"DeviceName": "Office5_BAlight",
                           "value": "False"},
                          {"DeviceName": "Office5_BWindow",
                           "value": "True"}
                         ]
          }
         ]
}
```


Figure. 3 User A IoT automation app as described in scenario 1

important effect in detecting conflicts.

5. Uncovering conflicts that can result from joint behavior situations, minimize their conflict coverage.

The proposed prototype helps to fill these gaps in recent IoT app verification frameworks. Also, it can be suitable for the intrinsically dynamic and unpredictable nature of IoT systems, thereby ensuring more system correctness and safety in building automation.

4. Methodology

In this section, we explain in detail the joint-behavior concept and the enhanced framework for conflict detection and resolution.

4.1 IoT apps interactions levels

In BAS, there are two main types of IoT apps which represent the automation behavior s [15, 35]. The first IoT app type is the user automation service (a.k.a. user policy or dynamic behavior policy) which represents user intentions to automate and control the surrounded environment behavior. The second IoT app type is the system policy (a.k.a. system axiom policy) which means invariants that must be valid and not allowed for any user's service to disregard.

According to the interaction detection taxonomy provided in [35], there are three types of interactions between these IoT apps types in the smart home as an example of smart buildings:

- Interactions between two system axiom policies,
- Interactions between a dynamic behavior policy and a system axiom policy, and
- Interactions between two dynamic behavior policies.

Similarly, Ibrhim et al. [15] proposed an IoT apps conflicts classification that defines the conflicts related to these interactions and mentioned another interaction between IoT apps that may take place in BAS or smart homes that is:

- Interactions between a set of IoT apps and system policies.

Following are definitions of some terms that emphasize the primary conflict type under discussion in this work.

Joint-behavior situation *a system situation, Sys_{sit}, where a set of IoT automation apps IoT_{apps} = {app_1, app_2, ..., app_n} satisfy these conditions:*

1. each $app_i \in IoT_{apps}$ is satisfied or triggered by Sys_{sit},
2. IoT_{apps} are belonging to different users,
3. IoT_{apps} sharing spatial-temporal aspects, and
4. individually, each IoT app in IoT_{apps} does not cause any type of conflicts according to [42].

The IoT apps in Figs. 3 and 4 represent the users' preferences as described in scenario 1. Based on the above definition, these IoT apps satisfy their conditions, as they have different owners (UserID =

```
{
  "ServiceID": 2,
  "UserID": 456,
  "UserPriority": 3,
  "StartDate": "2022-1-1",
  "Time": "11-00-00",
  "Period": 60,
  "RepeatEvery": ["Su"],
  "EndDate": "2022-2-1",
  "Rules":[
          {
           "Ser_ID": 2,
           "R_ID": 1,
           "Priority": 1,
           "RLoc": "Office5",
           "conditionGroup":[{"DeviceName": "Office5_BBoccupancy",
                              "value": "True"}
                            ],
           "actionGroup":[{"DeviceName": "Office5_BBlight",
                           "value": "True"},
                          {"DeviceName": "Office5_BAC",
                           "value": "True"}
                         ]
          }
         ]
}
```


Figure. 4 User B IoT automation app as described in scenario 1


```
{
  "Local checking": "No conflicts",
  "Global checking": [
     {
      "Against Policies": [
           {"Shared Policies": "Yes",
            "Conflicts": "No conflicts"} ],
      "Against Other IoT apps": [
           {"Shared IoT apps": "Yes",
            "Conflicts": "No conflicts"} ]
     } ]
}
```


Figure. 5 Conflict report generated by [42] when individually checking IoT apps in Figs. 3 and 4


```
{
  "Policy_ID": 1,
  "UserID": 1,
  "UserPriority": 1,
  "Constraints":[
      {
       "P_ID": 1,
       "PLoc": "Office5",
       "operatorGroup":
          {
           "deviceGroup": [
              {"DeviceName": "Office5_BAC",
               "value": "True"},
              {"DeviceName": "Office5_Bwindow",
               "value": "True"}
                          ],
           "Operator": "&",
           "Nigative": "!"
          }
      } ]
}
```


Figure. 6 System policy as described in scenario 1

123 and UserID = 456), and they overlap in time and location. Figure 5 represents the conflict report generated by [42] when checking these IoT apps individually either against shared services or against shared policies.

IoT apps joint-behavior conflict *based on [15], is the conflict that occurs when a joint-behavior situation takes place, and the effects of its IoT apps violate a system policy or a set of policies.*

Based on this definition, the joint-behavior situation that occurs between IoT apps in Figs. 3 and 4 violates the system policy in Fig. 6 since there is no satisfying assignment for,

$$(ALight = False \cap Window = True) \cap (BLight = True \cap AC = True) \cap \neg(Window = True \cap AC = True)$$

4.2 Joint-behavior conflict detection and resolution

One of the popular methods used to examine every possible state of the system is model checking [43]. Among the techniques that are still used to

Algorithm 1: Detection the Joint behavior conflicts

Input: a new *IoTapp* and the list of its locations *Locs*
Output: List of the joint-behavior conflicted IoT apps *Jconflist*

```
1  Jconflist := {}
2  foreach loc_i in Locs do
       /* Rules preparation                                   */
3      Overlaps = getOverlappedIoTapp(loc_i)
4      Combs = getAllCombinations(IoTapp, Overlaps)
5      foreach subset_j in Combs do
           /* Rules minimization                              */
6          if IoTapp ⊂ subset_j then
7              if subset_j ∉ Jconflist then
                   /* Policies preparation &
                      translation                             */
8                  Locpols = getAllPolicies(loc_i)
9                  trans_subset = translate(subset_j)
10                 foreach pol_k in Locpols do
11                     trans_pol = translate(pol_k)
                       /* Z3 checking                         */
12                     if (trans_subset ∩ trans_pol) is
                        UNSAT then
13                         add (trans_subset, trans_pol) in
                           Jconflist
```

determine the satisfiability for multi-theory logic formulas using a mathematical model is Satisfiability Modulo Theories (SMT) using SMT-based model checkers such as Z3 [44].

A step forward to make IoT apps and system policy in Figs. 3, 4, and 6 convenient to SMT model checkers, is to translate them to a standard logic language. In our proposed model, the SMT-based Z3 solver [44] and the SMTLib-v2 [45] standard language are used. In particular, joint-behavior IoT apps are translated to conditional assertions using the assert operator in prefix notation with => operator, which means a conditional statement. Unlike IoT apps, policies translated to fact assertions (i.e., without if clause) that are always true. For instance, the translation of scenario 1 IoT apps and policy mentioned previously are as follows:

User A IoT app:
(define-fun Aoccupany () Bool)
(define-fun Alight () Bool)
(define-fun window () Bool)
(assert (=> Aoccupany (and (not Alight) window)))

User B IoT app:
(define-fun Boccupany () Bool)
(define-fun Blight () Bool)
(define-fun AC () Bool)
(assert (=> Boccupany (and Blight AC)))
Location Policy:
(define-fun window () Bool)
(define-fun AC () Bool)
(assert (not (and window AC)))

4.2.1. Conflict detection

Pseudocode for the conflict detection process is provided in Algorithm 1.

Pseudocode analysis:

1. **Initialization (Line 1):** declares one variable, *Jconflist*, to store the set of rules and policies IDs that cause a Joint-Behavior conflict.
2. **Detection process (Lines 2-13):** this part describes the process used in identifying and detecting the conflict under consideration as follows:

   - *Rules preparation, (lines 3 & 4):* the overlapped rules are collected and stored in the *Overlaps* variable using the *getOverlappedIoTapp()* method by checking the time and locations of the input IoT app with the existing IoT apps. The method *getAllCombinations()* is used to generate all k-subsets of combinations between the rules in Overlaps without ordering. These k-subsets are stored in the *Combs* variable. The formula below is used to obtain the number of different possible combinations ${}_n\text{Combs}_k$.

$$ {}_n Combs_k = n! / (k! (n-k)!) \quad (1) $$

   For instance, let Overlaps includes four overlapped rules $\{R_B, R_C, R_D, R_E\}$ with RA of the new-added IoT app for a specific location. The number of rules used to generate combinations is 5. The 2-subset combinations where the number of rules in each subset is k = 2 is ${}_5\text{Combs}_2 = 10$ and includes these pairs of rules $\{R_AR_B, R_AR_C, R_AR_D, R_AR_E, R_BR_C, R_BR_D, R_BR_E, R_CR_D, R_CR_E, R_DR_E\}$.
   - *Rules minimization, (lines 6 and 7):* to minimize the number of combinations to be checked, only the combinations that contain rules of the new-added IoT are selected (line 6). According to the above example, only the subsets that contain R_A are selected for checking, which are $\{R_AR_B, R_AR_C, R_AR_D, R_AR_E\}$. Another minimization was performed to reduce the overhead load in the Z3 solver by ignoring any upcoming subset where some of its components are causing a joint-behavior conflict in a previous less sized subset (line 7). For instance, if the subset $\{R_AR_B\}$ causes a conflict, then all subsets of other combination sized that contain this pair of rules (e.g., $\{R_AR_BR_C\}$) are ignored, as it will generate the same conflict
   - *Policies preparation & translation, (lines 8 and 9):* the joint-behavior conflict is detected when the joint-behavior situation (defined in section 4) violates a system policy, for this, the policies constraints located in the same location are collected using *getAllPolicies()* method and stored in *Locpols* variable for further use. The *translate()* method is used to convert the rules or policies to an intermediate format (i.e., SMTLib syntax) as explained in subsection 4.2.
   - *Z3 checking, (lines 12 and 13):* to confirm that the conflict has occurred, the two translated rule subset and policy constraint, trans subset and trans pol respectively, are checked using the Z3 solver. If the intersection between them is UNSAT then

```
{
  "Conflict Level": "IoT apps' conflicts",
  "IoT app": 2,
  "Conflict Type": "Global IoT app with others against policy conflicts",
  "Conflicts": [
      {
       "Joint-behavior Conflicts": [
            {"First": 1,
             "Joint IoT apps": [
              {"IoT app ID": 1,
               "RID": 1}
             ],
             "With Policies":[
                {"PolicyID": 1,
                 "CID": 1}
             ],
             "Resolution":[
              {"DeviceName": "AC",
               "value": "False"}
             ]
            }
       ]
      }
  ]
}
```


Figure. 7 Conflict report generated by the conflict resolution for scenario 1 in Fig. 1


```
{"$schema": "http://json-schema.org/draft-04/schema#",
 "type": "object",
 "properties": {
 "ServiceID": { "type": "integer" },
 "UserID": { "type": "integer" },
 "UserPriority": { "type": "integer" },
 "StartDate": { "type": "string" },
 "Time": { "type": "string" },
 "Period": { "type": "integer" },
 "RepeatEvery": { "type": "array",  "items": [ { "type": "string" } ] },
 "EndDate": { "type": "string" },
 "Rules": { "type": "array",
    "items": [ { "type": "object",
                 "properties": { "Ser_ID": { "type": "integer" },
                                 "R_ID": { "type": "integer" },
                                 "Priority": { "type": "integer" },
                                 "RLoc": { "type": "string" },
                                 "conditionGroup": { "type": "array",
                                    "items": [ { "type": "object",
                                       "properties": {"DeviceName": { "type": "string" },
                                          "operation": { "type": "string" },
                                "value": { "type": "string" } },
                                                    "required": [ "DeviceName", "operation", "value" ] } ] },
                                 "actionGroup": { "type": "array",
                                    "items": [ { "type": "object",
                                                 "properties": { "DeviceName": { "type": "string" },
                                                                 "value": { "type": "string" } },
                                                 "required": [ "DeviceName", "value" ] } ] } },

                 "required": [ "Ser_ID", "R_ID", "Priority", "RLoc",
                               "conditionGroup", "actionGroup" ] } ] } },

    "required": [ "ServiceID", "UserID", "UserPriority",
                  "StartDate", "Time", "Period", "RepeatEvery", "EndDate", "Rules" ]
}
```


Figure. 8 The JSON schema used to generate IoT apps

the pair is added to the conflict list *Jconflist*.

3. **Time complexity:** Algorithm 1 iterates over the generated combinations and policies for each location in the submitted IoT app. Although the number of combinations and policies can be large for real-world smart buildings like large campuses, the algorithm only considers the rules and policies that occur in the same fine-grained location and follows a filtering process that considerably reduces the overall number of iterations and checking process. Under the assumption that the total number of locations customized by an IoT app is at most one location, algorithm 1 runs in $O(nm)$, where n is the number of combinations and m is the number of policies.

4.2.2. Conflict resolution

The next step is to generate a solution for the detected conflicts.

The joint-behavior conflict is solved using user prioritization, which focuses on the highest authority assigned to users [46]. Using prioritization, users are assigned a priority (given to them by the system

admin when adding a new user to the system) that represents the level of the user's capability to perform his IoT automation app over other users. For instance, in Figs. 3 and 4, the priorities of userA and userB are 1 and 3, respectively. According to this resolution method, the IoT app that will take place and change the environment states is the userB' IoT app.

Another resolution for the joint-behavior conflict is conducted by suggesting updating the new-added IoT app rules. Using Algorithm 1, the policy violated by the joint-behavior IoT apps can be determined. Using this policy, the valid values for the new-added IoT app devices that cause the joint-behavior conflict are suggested as a solution. Fig. 7 shows a conflict report generated from the conflict resolution when checking the joint-behavior for scenario 1 in Fig. 1. The conflict report shows the IoT apps (services with IDs 1 and 2) and the policy (policy ID = 1) causing the joint-behavior conflict. According to this resolution method, with the assumption that the service with ID = 2 is the new-add IoT app, the suggested resolution for the detected conflict is updating the value of the "AC" device to "False" in the service with ID = 2.

5. Results and discussion

This section gives a detailed overview of the conducted experiments as a proof-of-concept evaluation using a refinement dataset for the proposed IoT app joint-behavior conflicts checking.

5.1 Data refinement and setup

A proof-of-concept evaluation based on a dataset of hand-refinement IoT automation apps and policies collected from recent literature on IoT safety and security [47, 36, 39, 16, 38, 32] is performed.

The dataset includes 80 different IoT apps for 15 different locations. The total number of rules in the dataset is 117 rules. Also, the dataset consists of system policies with 17 constraints. The dataset is available online in a public Github repository[1] for further use. The dataset refinement process is as follows:

1- Customizing the domain of scenarios to cover diverse campus automation real-life use cases.
2- Adding the missing attributes for the IoT apps and policies such as location, time, and priority.
3- Configuring the scenarios to cause some joint-behavior violations as defined before.

The above synthetic refinement and the randomly generated IoT apps are represented in the JSON schema, Figs. 3 and 4 are examples of IoT apps using this JSON schema. The code used for generating the synthetic rules is uploaded to the same public GitHub repository mentioned before. The synthetic refinement code could be used to generate different types of rules (e.g., simple condition-action rules and complex rules) and policy assertions (e.g., device group conditions with the and/or operator). All the IoT apps and policies generated have the required environmental context attributes (time and location).

Fig. 8 shows an example of a schema that defines a simple IoT app. The first line contains information about the schema, $schema keyword, that defines the version of the schema follows (i.e., 'draft-04'). The keywords from lines 4 to 12 describe the IoT app metadata (i.e., IDs, time, location, and rules). The values of these keywords can be one of the supported JSON types or subtypes.

For instance, the Rules keyword is a JSON array type that contains metadata about the IoT app automation rules in the form of IFTTT-style rules. Each item in the rules array is an object that includes the required keywords to define the rule properties (i.e., Ser_ID, R_ID, Priority, RLoc, conditionGroup, and actionGroup). All the defined keywords in the JSON schema are required keywords, which requires each IoT app to have values for them.

The host machine was running Linux Mint 18.04 on hardware consisting of a Core i7 processor and 8GB of RAM. All evaluations are performed locally in the host machine using Eclipse IDE for running the code and curl commands[2] for requests to check the IoT automation apps.

Table 1 represents a subset of policies used to evaluate the proposed checking model. These policies correspond to real-world invariants used to control a building. For instance, policy4 ensures safety at the office entrance and says that the main office door should be locked when no one is in the office. Another policy, policy10, ensures occupants' safety in emergencies. If a joint-behavior situation violates any co-location policies, we conclude with a joint-behavior conflict.

5.2 Experiments

A theoretical comparison against a set of related work for supporting the joint-behavior conflict

[1] https://github.com/HamadaIbrahim-fci/automationservicesdataset, accessed on July 16, 2022.

[2] Command line tool and library for transferring data with URLs, https://curl.se/, August 5, 2022

Table 1. A subset of policies constraints used in the evaluation

ID	Location	Constraints
policy1	Office5	AC and Window must not be on at the same time
policy2	Hall1	Curtain and Window must be off when Datashow is running
policy3	Lab1	Temperature should be within a predefined range when students exist
policy4	Office3	The main door should be locked when no one is in office
policy5	Office2	Location mode should be changed to Away when no one is in office
policy6	Office1	An alarm should strobe/siren when detecting smoke in office
policy7	Office5	Some devices should not be turned on when no one is at office
policy8	Office4	The light should be off when no one is in hall
policy9	Hall3	The battery of devices must not be below a specified threshold
policy10	Hall2	The emergency alarming system must be on

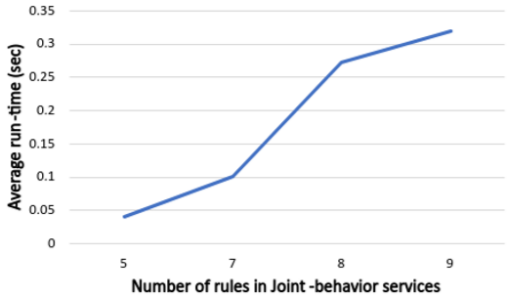

Figure. 9 Average run-time for joint behavior conflict

checking automation services interaction checking is shown in Table 2. Although all mentioned work in the comparison supports different users in creating IoT automation apps and supporting inter-IoT apps interactions checking, the proposed conflict checking has more features than previous work. The proposed conflict checking supports policy involvement and detects a joint-behavior conflict. The inter-IoT app interferences mean conflicts resulting from multiple IoT apps interplaying over shared devices under specific circumstances. The proposed works in [36, 38, 16, 42] detected IoT apps conflicts based on event chain between apps, specific device types (On/Off), or contradicting events. Works in [29, 23, 24, 32] detected IoT apps interferences based on device influences over specific environment entities (e.g., temperature).

For example, the scenario in Table 3 includes IoT

Table 2. Comparing the proposed joint-behavior IoT apps conflicts with other conflict detection frameworks

Reference	Features			
	Multiple Users	Inter-IoT apps interferences	Policies properties	Joint-behavior conflict detection
Nguyen et al. [36]	✔	✔	✔	✗
Wang et al. [29]	✔	✔	✗	✗
Liu et al. [38]	✔	✔	✔	✗
Xiao et al. [23]	✔	✔	✗	✗
Celik et al. [16]	✔	✔	✔	✗
Chaki et al. [24]	✔	✔	✗	✗
Gu et al. [32]	✔	✔	✗	✗
Ibrhim et al. [42]	✔	✔	✔	✗
The proposed conflict checking	✔	✔	✔	✔

app23 as the new-added IoT app, it overlapped in location and time with different IoT apps (e.g., IoT app1, IoT app5, IoT app18, IoT app22, IoT app26, and IoT app27). The total number of rules in these IoT apps is eight rules. To determine the IoT app(s) that cause a joint-behavior conflict with the IoT app23, all the interactions between this new-added IoT app and these IoT apps are checked by finding all possible combinations between them.

In the case of finding the combinations containing pairs of rules, k = 2, there are 36 combinations. This number of combinations signifies the number of the joint-behavior detection process. To optimize the detection process, this number is minimized by removing all combinations that do not include the IoT app23' rule. As a result of this minimization, there are only eight combinations. Each subset in these eight combinations is checked against all policy constraints that have the exact location to determine if it causes a joint-behavior conflict or not. Two joint-behavior conflicts between two pairs of IoT apps (IoT app23, IoT app22) and (IoT app23, IoT app27) are obtained.

With increasing the number of rules in the combination to 3 rules, k = 3, the joint-behavior detection process is performed for only nine subsets instead of checking all the 84 combinations. All

Table 3. A joint-behavior conflict scenario

New-added IoT app	Joint-behavior situation	Violated policy
IoT app23: BOccupancy = T → BLight = T AC = T	**IoT app1:** Userstay >= 30 → WFO = T WFO = T → Textstate = F **IoT app5:** Intemperature >=70 → AC = T Intemperature <=60 → Heater = T **IoT app18:** Smoke = T → Fan = T **IoT app22:** AOccupancy = T → ALight = F and Window = T **IoT app26:** Inoffice = T → AC = T **IoT app27:** CO2 > 1000 → Window = T	**Policy1:** AC and Window must not be on at the same time

combinations that include either (IoT app23, IoT app22) or (IoT app23, IoT app27) rule pairs are ignored as a second minimization to the number of the detection process. This process of creating combinations and minimizing the number of the detection process is followed until only one combination of size k = 9, which contains all IoT apps' rules is reached.

Fig. 9 shows the average run-time for checking joint-behavior conflict for some formulated scenarios. Here, the run-time is the time to detect and resolve the joint-behavior conflict. It includes collecting IoT apps' meta-data, files accessing and creating, and the time of converting IoT apps to intermediate formats. For each subset, the average run-time of ten conflict-checking processes using the Z3 solver is measured.

Also, Fig. 9 provides a proof-of-concept evaluation for the joint-behavior conflict detection. The average run-time for a scenario includes four overlapped rules, which is 0.04 sec is much less than the average run-time taken by a different scenario includes eight overlapped rules of 0.319 sec. Some factors causing this significant difference in terms of average run-time among them are the number of overlapped rules in the joint-behavior scenario, which determines the combinations produced, the number of policy constraints involved in the checking process, and the number of ignored ones services during the checking process.

6. Conclusion and future work

This paper proposed a simple yet effective conflict detection and resolution process for the conflict resulting from violating system or environment policies due to multi-user contexts overlapping. The proposed checking process considers an extension of the framework provided in [42] and is also based on SMT model checker Z3 and its related SMTLib-v2 formalization language. The main advantage of the proposed checking process is that it can be integrated and used in different BAS verification systems to increase the system's degree of safety and correctness. We conducted a proof-of-concept experiment based on a dataset collected and refined from related work. Based on our sought, the joint-behavior conflicts are correctly detected and resolved against all policy constraints in the dataset. Also, results are promising in detecting the conflict between the different number of IoT apps in a reasonable average run time. Among the future works needed are added other uncovered conflicts such as time-based conflicts that result in temporal behavior of rules and enhancing the framework to support large-scale industrial building in the IoT automation domain.

Funding

This research is funded by the Deanship of Scientific Research, Islamic University of Madinah, Madinah, Saudi Arabia.

Conflicts of interest

The authors declare no conflict of interest.

Author contributions

"Conceptualization, Hamada, Hisham, and Emad; methodology, Hamada, Ahmed and Emad; Implementation, Hamada; Hisham, Ahmed, and Emad; formal analysis, Hamada; Hisham, Ahmed, and Emad; writing—original draft preparation, Hamada; Hisham, Ahmed, and Emad; supervision, Emad and Hisham".

References

[1] P. Stluka, G. Parthasarathy, S. Gabel, and T. Samad, "Architectures and Algorithms for

Building Automation--an Industry View", *Intelligent Building Control Systems*, Springer, pp. 11-43, 2018.

[2] S. A. Karvigh, A. Ghahramani, B. B. Gerber, and L. Soibelman, "One size does not fit all: Understanding User Preferences for Building Automation Systems", *Energy and Buildings*, Vol. 145, pp. 163-173, 2017.

[3] D. H. Ford, "Humans as Self-constructing Living Systems: A Developmental Perspective on Behavior and Personality", *Routledge*, 2019.

[4] M. Mangla, R. Akhare, and S. Ambarkar, "Context-aware Automation Based Energy Conservation Techniques for IoT Ecosystem", *Energy Conservation for IoT Devices*, pp. 129-153, 2019.

[5] P. Domingues, P. Carreira, R. Vieira, and W. Kastner, "Building Automation Systems: Concepts and Technology Review", *Computer Standards & Interfaces*, Vol. 45, pp. 1-12, 2016.

[6] J. D. L. Morenas, C. M. D. Silva, J. Barbosa, and P. Leitao, "Low Cost Integration of IoT Technologies for Building Automation", In: *Proc. of IECON 2019-45th Annual Conference of the IEEE Industrial Electronics Society*, Vol. 1, pp. 2548-2553, 2019.

[7] B. Ur, E. M. Manus, M. P. Y. Ho, and M. L. Littman, "Practical Trigger-Action Programming in the Smart Home", In: *Proc. of the SIGCHI Conf. on Human Factors in Computing Systems, ACM*, pp. 803-812, 2014.

[8] J. Huang and M. Cakmak, "Supporting Mental Model Accuracy in Trigger-Action Programming", In: *Proc. of the 2015 ACM International Joint Conference on Pervasive and Ubiquitous Computing*, pp. 215-225, 2015.

[9] M. Funk, L. Chen, S. Yang, and Y. Chen, "Addressing the Need to Capture Scenarios, Intentions and Preferences: Interactive Intentional Programming in the Smart Home", *International Journal of Design*, Vol. 12, No. 1, pp. 53-66, 2018.

[10] Z. Pan, S. Hariri, and J. Pacheco, "Context Aware Intrusion Detection for Building Automation Systems", *Computers & Security*, Vol. 85, pp. 181-201, 2019.

[11] M. Jain, A. Varma, N. Kaushik, and K. Jayavel, "Building Automation and Context Aware Energy Consumption using IoT—Smart Campus", *International Journal of Applied Engineering Research*, Vol. 12, No. 14, pp. 4213-4220, 2017.

[12] R. Mohamed, T. Perumal, M. Sulaiman, and N. Mustapha, "Multi Resident Complex Activity Recognition in Smart Home: a Literature Review", *International Journal of Smart Home*, Vol. 11, No. 6, pp. 21-32, 2017.

[13] R. Garg and C. Moreno, "Understanding Motivators, Constraints, and Practices of Sharing Internet of Things", In: *Proc. of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, ACM*, Vol. 3, No. 44, pp. 1-21, 2019.

[14] C. Geeng and F. Roesner, "Who's in Control? Interactions in Multi-user Smart Homes", In: *Proc. of the 2019 CHI Conf. on Human Factors in Computing Systems, ACM*, No. 268, pp. 1-13, 2019.

[15] H. Ibrhim, H. Hassan, and E. Nabil, "A Conflicts' Classification for IoT-based Services: a Comparative Survey", *PeerJ Computer Science*, Vol. 7, p. e480, 2021.

[16] Z. Celik, G. Tan, and P. D. M. Daniel, "IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT", In: *Proc. of the Network and Distributed System Security (NDSS) Symposium*, 2019.

[17] C. Nandi and M. D. Ernst, "Automatic Trigger Generation for Rule-based Smart Homes", In: *Proc. of the 2016 ACM Workshop on Programming Languages and Analysis for Security*, pp. 97-102, 2016.

[18] S. Yarosh and P. Zave, "Locked or not? Mental Models of IoT Feature Interaction", In: *Proc. of the 2017 CHI Conf. on Human Factors in Computing Systems*, pp. 2993-2997, 2017.

[19] E. Zeng, S. Mare, and F. Roesner, "End User Security and Privacy Concerns with Smart Homes", In: *Proc. of the Thirteenth Symposium on Usable Privacy and Security (SOUPS 2017)*, pp. 65-80, 2017.

[20] R. Jayaprakash, S. Nagarathinam, S. Agrawal, R. Suriyanarayanan, and A. Sivasubramaniam, *System and Method for Efficient Verification of Building Automation Systems*, U.S. Patent, 2021.

[21] E. Zeng and F. Roesner, "Understanding and Improving Security and Privacy in Multi-User Smart Homes: A Design Exploration and In-Home User Study", In: *Proc. of the 28th USENIX Security Symposium (USENIX Security 19)*, pp. 159-176, 2019.

[22] C. Barrett and C. Tinelli, "Satisfiability Modulo Theories", *Handbook of Model Checking*, *Springer*, pp. 305-343, 2018.

[23] D. Xiao, Q. Wang, M. Cai, Z. Zhu, and W. Zhao, "A3ID: an Automatic and Interpretable Implicit Interference Detection Method for Smart Home via Knowledge Graph", *IEEE Internet of Things Journal*, Vol. 7, No. 3, pp. 2197-2211, 2019.

[24] D. Chaki, A. Bouguettaya, and S. Mistry, "A

Conflict Detection Framework for IoT Services in Multi-resident Smart Homes", In: *Proc. of 2020 IEEE International Conf. on Web Services (ICWS)*, pp. 224-231, 2020.

[25] A. A. Farooq, E. A. Shaer, T. Moyer, and K. Kant, "IoTC 2: A Formal Method Approach for Detecting Conflicts in Large Scale IoT Systems", In: *Proc. of the 2019 IFIP/IEEE Symposium on Integrated Network and Service Management (IM)*, pp. 442-447, 2019.

[26] M. Palekar, E. Fernandes, and F. Roesner, "Analysis of the Susceptibility of Smart Home Programming Interfaces to End User Error", In: *Proc. of the 2019 IEEE Security and Privacy Workshops (SPW)*, pp. 138-143, 2019.

[27] T. Shah, S. Trusit, T. Ngo, and K. Neelamegam, "Conflict Detection in Rule Based IoT Systems", In: *Proc. of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conf. (IEMCON)*, pp. 0276-0284, 2019.

[28] A. Krishna, M. L. Pallec, R. Mateescu, L. Noirie, and G. Salaün, "IoT Composer: Composition and Deployment of IoT Applications", In: *Proc. of the 2019 IEEE/ACM 41st International Conf. on Software Engineering: Companion Proceedings (ICSE-Companion)*, pp. 19-22, 2019.

[29] Q. Wang, P. Datta, W. Yang, S. Liu, A. Bates, and C. A. Gunter, "Charting the Attack Surface of Trigger-Action IoT Platforms", In: *Proc. of the 2019 ACM SIGSAC Conf. on Computer and Communications Security*, pp. 1439-1453, 2019.

[30] T. Melissaris, K. Shaw, and M. Martonosi, "OKAPI: in Support of Application Correctness in Smart Home Environments", In: *Proc. of the 2019 Fourth International Conf. on Fog and Mobile Edge Computing (FMEC)*, pp. 173-180, 2019.

[31] Y. Lee and F. J. Lin, "Situation Awareness and Conflict Resolution in Smart Home with Multiple Users", In: *Proc. of the 2019 IEEE 5th World Forum on Internet of Things (WF-IoT)*, pp. 852-857, 2019.

[32] T. Gu, Z. Fang, A. Abhishek, H. Fu, P. Hu, and P. Mohapatra, "Iotgaze: IoT Security Enforcement via Wireless Context Analysis", In: *Proc. of the IEEE INFOCOM 2020-IEEE Conf. on Computer Communications*, pp. 884-893, 2020.

[33] S. Han, G. M. Lee, and N. Crespi, "Towards Automated Service Composition using Policy Ontology in Building Automation System", In: *Proc. of the 2012 IEEE Ninth International Conf. on Services Computing*, pp. 685-686, 2012.

[34] S. Han, G. M. Lee, and N. Crespi, "Context-aware Service Composition Framework in Web-enabled Building Automation System", In: *Proc. of the 2012 16th International Conf. on Intelligence in Next Generation Networks*, pp. 128-133, 2012.

[35] M. Shehata, A. Eberlein, and A. Fapojuwo, "Using Semi-formal Methods for Detecting Interactions among Smart Homes Policies", *Science of Computer Programming*, Vol. 67, No. 2-3, pp. 125-161, 2007.

[36] D. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M. Colbert, and P. M. Daniel, "IotSan: Fortifying the Safety of IoT Systems", In: *Proc. of the 14th International Conf. on emerging Networking Experiments and Technologies*, pp. 191-203, 2018.

[37] G. Holzmann, "The Model Checker SPIN", *IEEE Transactions on Software Engineering*, Vol. 23, No. 5, pp. 279-295, 1997.

[38] R. Liu, Z. Wang, L. Garcia, and M. Srivastava, "RemedioT: Remedial Actions for Internet-of-Things Conflicts", In: *Proc. of the 6th ACM International Conf. on Systems for Energy-Efficient Buildings, Cities, and Transportation*, pp. 101-110, 2019.

[39] Z. Celik, P. M. Daniel, and G. Tan, "Soteria: Automated IoT Safety and Security Analysis", In: *Proc. of the 2018 USENIX Annual Technical Conf, (USENIX ATC 18)*, pp. 147-158, 2018.

[40] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella, "Nusmv 2: An Opensource Tool for Symbolic Model Checking", In: *Proc. of International Conf. on Computer Aided Verification*, pp. 359-364, 2002.

[41] M. Balliu, M. Merro, M. Pasqua, and M. Shcherbakov, "Friendly Fire: Cross-app Interactions in IoT Platforms", *ACM Transactions on Privacy and Security (TOPS)*, Vol. 24, No. 3, pp. 1-40, 2021.

[42] H. Ibrhim, S. Khattab, K. Elsayed, A. Badr, and E. Nabil, "A Formal Methods-based Rule Verification Framework for End-user Programming in Campus Building Automation Systems", *Building and Environment*, Vol. 181, 2020.

[43] R. Jhala and R. Majumdar, "Software Model Checking", *ACM Computing Surveys (CSUR)*, Vol. 41, No. 4, pp. 1-54, 2009.

[44] L. Moura and N. Bjørner, "Z3: An Efficient SMT Solver", In: *Proc. of International Conf. on Tools and Algorithms for the Construction and Analysis of Systems*, pp. 337-340, 2008.

[45] C. Barrett, A. Stump, and C. Tinelli, "The Smt-

lib Standard: Version 2.0", In: *Proc. of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, Vol. 13, pp. 14, 2010.
[46] J. Durkin and J. Durkin, "Expert systems: design and development", *Prentice Hall PTR*, 1998.
[47] W. Ding and H. Hu, "On the Safety of IoT Device Physical Interaction Control", In: *Proc. of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, pp. 832-846, 2018.