IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 23 April 2024, accepted 5 June 2024, date of publication 17 June 2024, date of current version 24 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3415632

RESEARCH ARTICLE

Ontology-Based Classification and Detection of the Smart Home Automation Rules Conflicts

ADEEB MANSOOR ANSARI, MOHAMMED NAZIR, AND KHURRAM MUSTAFA
Department of Computer Science, Jamia Millia Islamia, New Delhi, India
Corresponding author: Adeeb Mansoor Ansari (adeebmansooransari.am@gmail.com)

ABSTRACT Smart homes are the most adaptable and utilized application of the IoT. Smart homes enable end users to define and control the automation remotely by defining automation rules. The automation rules interaction results in the defined tasks and activities but also delivers unwanted interactions, which show adverse effects. Their interactions may consist of chained and covert rules interaction, which are hard to trace and identify. Attackers' malicious apps installed in smart homes may leverage rules' adverse effects to compromise smart home security and harm users, such as opening the window at night or when no one is home. To address such issues, we developed an ontology of smart home automation rules to identify such interactions. Prior works failed to provide the complete classification and could not identify all possible rule conflicts. In this work, we proposed the classification of automation rule interactions into five categories and recognized the potential rule conflicts. In addition, the ontology can examine rules interactions against the defined safety policies, providing an extra layer of security. The ontology will aid the designers in better understanding and developing a robust security mechanism at the design phase to resist rules interaction conflicts and their adverse effects. We examine this work on the thirty-five rules formed in the light of prior work test cases and validation. The results show that the proposed work is promising and can efficiently identify all the rule interaction conflicts in the smart home.

INDEX TERMS Smart home, ontology, automation rules, rule conflict, conflict classification, security.

I. INTRODUCTION

The adaptation of the Internet of Things is increasing day by day, connecting everything and existing everywhere, forming a vast network connected with the Internet [1], [2]. One of the prominent applications of the Internet of Things is the smart home, which enables its user's convenience and management of the smart home with a click [3], [4], [5]. Smart homes have sensors and actuators that alter the environment and actuators' states according to the user's defined automation to deliver comfort and convenience [2], [6]. The user-defined automation rules access and assign the value to the various variables and actuator states to achieve automation. When there are multiple automation rules for performing various activities; these rules may access and assign the same variables and actuator states and share the same space in the smart home [7], [8].

Rules sharing the same variables, states, and space interact with each other directly or indirectly [8], [9]. These interactions may cause conflicts, resulting in nondeterministic outcomes after rule execution [4], [5], [7], [10]. For example rule R1 states open the window when the temp is greater than 27° C, and R2 dictates close the window when the TV is on to enable a dark room. If R1 and R2 occur at the same time and location, then the window state outcome will be nondeterministic, i.e., the window can be opened or closed. Numerous automation rules for the same smart space make the rule's interactions more complex [9]. Further, in multi-resident smart homes, this complexity escalates with the multiple users' authority to define rules [2]. Such rules conflict may damage the devices, leverage them to perform other attacks, and lead the smart home to compromised states. This poses a significant threat to the users economically and in terms of their lives. Therefore, detecting the rules conflicts is essential to secure smart homes and their users [11].

Detection of rules conflicts has recently been a hot topic in smart home security. Various prior works proposed conflict

The associate editor coordinating the review of this manuscript and approving it for publication was Leandros Maglaras.

detection mechanisms based on detection methodology. These existing works can be classified in model checking approach [16], [18], [19], [20] and classification-based formal approach [12], [13], [14], [15], [21] detection. In model checking-based detection, by extracting the information from the defined rules and controlling apps, the model of the smart home is formed. Then, the model is checked for conflicting interactions and against the safety policies to validate the overall smart home security [16]. On the other hand, the classification-based formal approach detection identifies the possible interaction of rules and detects potential conflicts according to the classification categories.

In modeling-based detections, collecting the information from the automation apps and correctly modeling the interaction is challenging due to the closed-source apps and frameworks. In formal detection mechanisms, classification serves as the backbone of conflict detection, but existing conflict classification needs to be completed, i.e., it only covers some possible rule conflicts. Pertinent literature like, Ibrhim et al., [8] proposed the classification of rule conflicts, which covers all the existing rule conflicts and safety policy violations. They also considered the multiuser authority conditions in defining the conflicts and violations. Although they classify the conflicts extensively, they still leave out the same environment impact conflict. Moreover, instances of the one conflict type are categorized as distinct conflicts, which can be classified collectively as one conflict class. For example execution conflict and indirect conflict are instances of cross-execution conflict class. Both conflicts occur when opposite actions are performed on the same actuator, either directly or indirectly [5]. This result in complexity in understanding the rule conflicts and thus effect the identification of all possible conflicts.

While the proposed ontology covers all the rule conflicts that may occur in the smart home. Based on it, rule conflicts and safety policy violations are detected. Rule inconsistencies that occurred during the rules set up by the users, which are termed as errors, not as conflicts, are out of the scope of this work. Further, detection algorithms in existing works do not cover all rule interactions, i.e., the interaction of a rule with its chained rules (if they exist). Among the simultaneous executing rules, chained rules of one rule with the other rule and its chained rules. These interactions are complex to trace due to their covert nature but are essential to consider. These uncovered rule interactions may leave out possible rule conflicts. Therefore, defining the rule conflicts, developing a complete classification, and evaluating all the complex possible rule interactions are essential to develop an efficient conflict detection mechanism [8].

To develop a secure and trustworthy smart environment such as a smart home, the problem of automation rules conflicts detection and its resolution must be realized throughout its development lifecycle from design to implementation. Therefore, to better understand these conflicts at each development lifecycle phase, we developed an ontology that provides designers and implementers a semantic model of the relation between automation components, rule interactions, and conflicts. Ontologies are effective in representing the complex structure of automation rules, devices, their relationships and the roles of different components of the smart home [41], [42], [43]).

Ontological models help to develop a formal framework to represent the automation rules semantics of the smart home with clarity and precision, which facilitates accurate detection and classification of the conflicts [41], [42]. Such conflict detection can analyze the rule interactions by reducing the false positives and thus improving the overall accuracy of the conflict detection. Ontology-based models enable automated conflict detection, which alleviates the need for manual inspection of rule conflicts, effort, and time of smart home users, thus ensuring prompt resolution and minimal disruptions to the smart home environments [42]. Furthermore, the scalability and adaptability of ontology-based models can accommodate complex and different configurations of smart home environments, which ensures the reliability and efficiency of such conflict detection mechanisms [41], [43].

Secondly, ontology provides a shareable and extendable knowledge base for each development lifecycle phase that supports advancements and improvements at each level to develop an effective and efficient tool, platform, and application to classify and detect automation rules in a smart environment. In this paper, we proposed the ontology based smart home rule conflicts classification and detection mechanism to address the research gaps and limitations discussed above effectively. We proposed a rule conflict classification that covers all the possible rule conflicts and developed the ontology illustrating how to detect the rule conflicts efficiently. Following are the contributions of this research work.

1. Proposed the classification of rule conflicts that covers all the possible interaction conflicts. We defined the classes of conflicts based on the rules interaction adverse effect nor the occurrence situation of the conflict. Cross-execution conflict occurs when two rules command contradictory actions on the same actuator at the exact location. Either the rules command the contradictory actions directly or indirectly through the chaining of the rules. If the conditions resulting in adverse effects are the same and then they belong to the same conflict class.

2. Developed the ontology for the detection of smart home rule conflicts. The proposed detection method illustrates how to consider all the complex and covert rule interactions to detect all possible conflicts efficiently and effectively.

3. The proposed ontology and conflicts classification also detect the safety policy violations/policy constraints defined to guard the system. Per our understanding, this is the first formal representation that detects conflicts and policy constraint violations.

4. Developed an ontological mechanism that is expandable and shareable. This may be utilized at each development

phase of smart homes and further enhanced at each stage for the next development phase. Ontology is a formal method that gives the proper understanding for developing a robust and efficient mechanism.

The rest of the paper is organized as follows: Section II is related work, section III discusses the classification of the conflicts, section IV gives the background of smart home and ontology, section V explains the proposed ontology, section VI discusses the results and evaluation and lastly—the conclusion.

II. RELATED WORKS

In recent years, we have understood that conflicting automation rules are one of the highlighted problems in the smart home security domain. Identification of automation rule conflicts involves understanding the relation of automation components, classification of automation rules, and analysis of the effect/ impact of automation rules on smart space through environment and actuator states. In past years, promising works have been proposed to detect, classify, and resolve automation rule conflicts. Munir and Stankovic [38] proposed Depsys to detect and resolve conflicts based on emphasis, effect, and actuators and sensors dependencies but does not consider rule redundancy as the conflict. Whereas some work proposed the modeling of the smart home automation system by extracting rules and parameters from the app and validating against the functional and safety policies (including what should or should not occur by triggering a particular event) to check anomalies that lead the system to the unsafe state. Such as in SIFT [18] automation rules are translated into logical rules and checked by symbolic execution and model checking. In addition, safety policies are also checked for safety violations by the rules. While in Soteria [19], static analysis is performed to extract rules to form intermediate representation (IR) and then into a state model to check them against the safety policies to detect the anomalies in the defined automation [19]. Whereas in IoTGuard [20] instruments each app to extract its behavior and form a dynamic model that represents the combined runtime execution behavior of the apps before its execution. Then each app is checked against the defined safety policies and the report is sent to the app to block or pass the execution according to the security service response. On the other hand, IoTSAN [16] models the IoT system by taking the app's behavior, system configuration, and safety policies as input. When a new app is installed, IoTSAN validates the safety policies and reports the user for malicious apps, bad apps, and misconfigurations.

Other works, a formal framework of rule conflict detection and classification suggested by Sun et al. [12], not only detects conflicts at the execution phase but also verifies rules at the creation phase and develops a knowledge base to expedite conflict detection that improves efficiency. Sun et al. [13] performed the detection by storing rules featuring information in the database. Then, conflicts are detected through the proposed algorithms according to the defined rules' relation and conflict classification for each pair of rules. In addition, Chi et al. [21] coined the term CAI threats for cross-app interaction threats. Moreover, these CAI threats are due to the rules interaction with apps. Each app's rules are extracted and stored, whenever a new app is installed. Conflict detection is performed with the already installed apps, and results are shown to the users to decide. Lin et al. [14] proposed TAS (Trigger, Action, Status) classes to identify the potential redundancy wholly or partially. They checked new rule redundancy with existing rules before storing them in the database, which were discarded or merged accordingly, and reported to the user. TAS does not detect or classify the redundancy resulting from the chaining of the rules. In comparison, Wang et al., [15] describe the rule conflicts as inter-rule vulnerabilities and proposed iRuler to detect the threats of rule interaction. iRuler extracts the rules and forms the model comprising the IoT deployment system and information flow. Then, the model is checked using satisfiability modulo theories to detect inter-rule vulnerabilities. Shah et al. [5] focussed on the detection of incomplete rules (rules whose all values actions are not defined), for example, ''Rule 1: if the temperature is greater than 70 ◦F, turn on the air conditioner (incomplete). Rule 2: if the temperature is greater than 70 ◦F, turn on the air-conditioner, and when the temperature reaches 60 ◦F, turn off the air-conditioner (complete)''. In rule 1, termination of the rule is not defined. For conflict detection, they transform the rule in DNF (disjunctive normal form) and identify the overlapping DNF terms for each actuator. Further, incompleteness is resolved by defining termination action, i.e., the body of a rule is based on ''if'' (trigger) ''then'' (action). In addition, ''else'' is also included to assign an action for the unassigned values and to detect incomplete rules they use polygon problem [5]. For conflict resolution, the priority of the user is considered.

Further, some prior work proposed ontology-based classification and detection. Sang et al. proposed SPIDER [4], an ontology that classifies and detects the automation rules interaction adverse effects. However, SPIDER's proposed rules classification cannot detect all possible interaction effects. Camacho et al. [7] systematically defined the problem of conflict from scratch. The ontology they proposed focussed on selecting the combination of services in terms of comfort and energy efficiency. The knowledge base they proposed is fundamental and generic,and is not considered efficient automation conflict classification. Henceforth, it does not detects possible conflict between automation rules. Whereas, Rocher et al. [24] formed the ontology for DevOps development, where functionalities are unknown. They described only direct and indirect conflicts and did not elaborate on them in different or simplified categories. Their proposed ontology describes invoking the ACM (Actuator Conflict Management) module whenever a conflict is encountered. To resolve conflicts, they extracted the semantics of conflicting rules and fixed them according to the best way to fulfill both rules' objectives. While Chaki et al. [2]

used a hybrid approach utilizing ontological knowledge and a formal conflict model that defines rules conflict formally. They focussed on the service requirement sequence of all users to check the conflict, i.e., one requirement not clashing with another at the same time and in the exact location. They identified the overlapping services in a time interval at a location. Then, these overlapping services for each resident are checked for conflict by the ontology and formal model. On the other hand, Huang et al. [9] proposed a knowledge graph to detect conflicts and convert them into context-based graphs using contextual information about smart homes. They connect the relation of automation rules with the environment entities utilizing NLP techniques. They form the generalized knowledge base and convert it into contextual knowledge using the user-defined ECA (Event, Condition, Action) rules. Then, conflicts are detected by the proposed algorithm based on the relations inferences from the knowledge base.

The existing literature ranges from applicability to completeness, i.e., model checking-based work shows applicability, while classification-based frameworks show the efficiency and effectiveness of addressing the issue. Still, the existing methods and solutions persist in addressing some research gaps and challenges. In the current literature, the proposed classification of rule conflict needs to be completed, i.e., it only covers some possible conflicts that serve as the backbone for detection. Further, in the detection process, prior work considered the simultaneous or overlapping executing rules for conflict detection. Only a few works considered chained rule pairs for the detection of the conflicts. But one complex relation remains unnoticed for the detection: the relation of the simultaneously executing rules chained rules (if any) with the other rules and their resulting chaining. For example, if a, g, and p are simultaneously executing rules, each rule independently initiates the chaining $a \rightarrow b \rightarrow c \rightarrow d$, $g \rightarrow h \rightarrow i$ and $p \rightarrow q \rightarrow r$, then the relation of rules b, c, d with the g, h, i, p, q and r. Such relations are complex because simultaneous and chaining rules relations are transitive. Existing conflict detection schemes and methods lack consideration of such a relation. These research gaps motivate us to address such challenges and develop a robust, efficient and complete rule conflict detection mechanism.

III. CLASSIFICATION OF SMART HOME AUTOMATION RULES CONFLICTS

Detection of rule interaction adverse effects and conflicts requires appropriate understanding of how these rules interact and how can they be detected. Recent research showcases that the classification of rule interaction serves as the backbone to detect rule conflicts due to their implicit or explicit interaction. Many prior research proposed the classification of automation rules to detect them. Sun et al. [12] classified the automation rules based on device and environment impact after the rule's execution. They classified the conflicts into six categories based on an equivalent set and partial set of triggering conditions/events that cover rule redundancy, cross-execution conflict, and read-write and write-write conflict. As per our understanding, read-write conflict may refer to partial chaining (only through the environment parameters). However, they did not define write-write conflict as what condition or situation of rules interference it depicts. Chi et al. [21] (CAI) classify the rules interaction in action, triggers, and condition interference threats in seven categories that conclusively cover chaining, rules looping, cross execution, and cross environment impact interactions. Their proposed classification lacks rule redundancy and same environment impact.

Lin et al. [14] proposed TAS (Trigger, Action, Status) classes to identify the potential rule redundancies that are either partially or entirely redundant. TAS defines the rule redundancy in trigger types (triggered time, environment, status) and status types (sensor nodes, target status). However, TAS does not detect or classify the redundancy resulting from the chaining of the rules. While Wang et al. [15] describes the rule conflicts as inter-rule vulnerabilities and classify rule conflicts into six categories: condition bypass, condition block, action reverts, action conflict, action loop, and action duplicate. Where condition bypass depicts triggering without occurring the event, such conflicts are not due to the interaction of rules but by the user carelessness, i.e., defining one security-sensitive rule for an action occurring and defining other rules for the same action without considering security. Condition block shows suppressing the rule to execute via disabling the event. Such rule vulnerability is evident as each rule has some triggering conditions that must be true to trigger the rule. It is one of the adverse possibilities of chained rules for example rule R1 triggers rule R2 and rule R1 dictates to open the window, the triggering condition of the rule R3, but rule R2 states close the window. This changes the event state responsible for triggering rule R3 before its execution and leads to the condition block. Action revert illustrates that the action effect of one rule is revered by another rule action immediately. Action conflicts describe reaching a nondeterministic state after rule interaction. Action loop describes an action that triggers an event that triggers it again, and action duplicate reflects the repetition of the same action by different rules.

Sun et al. [13] defined eleven rules relations and on this relation basis, the conflicts were categorized into five classes. Shadow conflicts refer to redundancy, Execution Conflict (cross execution), Environment Mutual Conflict (Cross environment impact), Indirect Dependence Conflict (IDC), and direct dependence conflict (DDC). Where IDC and DDC are the rules looping, prior is looping due to the chaining of rules, and later is direct. Although [15] defined six categories and [13] proposed five categories of rule conflicts, both conclusively refer to the same classes: redundancy, rule looping, and cross-execution conflicts on actuator and environment. Shah et al. [5] proposed a generic and primary classification of rule conflict. Rule redundancy and cross-execution conflicts as independent and

execution conflicts in the cross-execution conflict category is considered. Huang et al. [9] proposed a classification of direct and indirect conflicts in 4 categories: function-function, cumulative environmental impact, transitive environment impact, and opposite environment impact conflicts that cover cross-execution conflict. They considered only chaining through the environment (partial chaining). As well as same and cross environment impact conflicts. However, rule redundancy is not considered in classifying the rule conflicts and chaining through the action trigger relation of actuators.

Chaki et al. [2] classify the rules conflict in terms of functional and non-functional attributes of the rules and Service impact conflicts in Direct and Indirect service impact. In addition, QoS attributes of services such as resource capacity (no. of requests executed by the actuator at a time), Qualitative (preference of the same service is different for different users at a time), and Quantitative (preference of a service is different for different users at the same time in terms of a numeric value like luminosity value) non-functional attributes are also considered for the conflicts. These non-functional attributes are the comfort preferences of the users for the same service at the same time. When different preferences occur, non-functional conflicts may lead to rule redundancy with distinct values. Functional conflicts cover cross-execution, direct service impact covers pure chaining (one rule depends on another for its execution), and indirect service impact reflects cross-environment impact conflict.

The definitions of the rule conflicts according to the proposed classification and detection is given below. Where, T: Triggers, C: Action commands, S: State of the actuator, and I: Impact on the environment after execution of the rules. Here T= $\bigwedge_{t=i}^{n} t$ (where $t_1, t_2, \ldots, t_n$ are triggering events), C=$\bigwedge_{c=i}^{n} c$ (where $c_1, c_2, c_3, \ldots, c_n$ are the action commands of the rules) and Rules $R = \{T, C, S, I\}$ is represented by the set of triggering events, action commands, the state change of the actuator, and the impact on the environment after execution of rules.

A. RULE REDUNDANCY CONFLICT

When different rules execute the same action on the same actuator simultaneously, the actuator repeats the action repeatedly. This may harm the device and discomfort the user. Repetition of the same action is not power efficient and economical.

Definition 1: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are simultaneously executing rules. $R_1 \rightarrow C_1$ and $R_2 \rightarrow C_2$ (Rule R_i commands C_i and C_i changes the state of the actuator to S_i); C_1 and C_2 are the action commands on the same actuator.

If $C_1 == C_2$ or $C_2 \subseteq C_1$, then R_1 and R_2 interaction results in rule redundancy conflict.

B. SIMULTANEOUS EXECUTING RULES

Triggering such rules may be due to the inclusion of triggers of one rule in another or the same set of triggers. They may be executed due to the same time event/environment parameter value range, or one rule triggering time event(or environment parameter) value range be in between or intersecting in another rule triggering range partially or wholly. Chaining rules are simultaneous to each other, and if R1 is simultaneous with R2, then the chained rules of R1 are simultaneous with R2 and its chained rules.

Definition 2: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are rules (Rule T_i triggers R_i and C_i changes the state of the actuator to S_i). Where, either T_1 and T_2 are overlapping events or states of the actuator. If $T_1 == T_2$ OR $T_1 \subseteq T_2$. Then, R_1 and R_2 are simultaneous executing rules.

C. CROSS-EXECUTION CONFLICT

When different rules request contrast action on the same actuator simultaneously. In other words, simultaneous rules execute contrast actions on the same actuator or device. This leads to the non-deterministic state of the actuator after execution of the conflicting rules, i.e., R1 commands ON to the actuator, and R2 commands OFF. Then, it is difficult to know the final state of the actuator; is it ON or OFF?

Definition 3: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are simultaneously executing rules. $R_1 \rightarrow C_1$ and $R_2 \rightarrow C_2$ (Rule R_i commands C_i and C_i changes the state of the actuator to S_i); C_1 and C_2 are executing on the same actuator. If $C_1 = \neg C_2$ or $C_2 = \neg C_1$, then R_1 and R_2 interaction results Cross execution conflict.

D. RULES LOOP

When the R1 action is the triggering event for the R2 rule, and the R2 rule action is the triggering event for the R1 rule, for example R1 triggers when the Air purifier is closed and commands window open. R2 triggers when the window is open and commands Air purifier close. R1 triggers R2, and R2 triggers R1 in a cycle. Rules Loop can be Direct and Indirect between the two rules. Direct loop rules occur when two rules trigger one another directly, as the described example. Indirect loop rules occur when loop rules trigger one another through a chain of rules like R1 triggers R2, R2 triggers R3, ……., Rn-1 triggers Rn, and Rn triggers R1; in this, R1 and Rn are indirect loop rules as they are connected through a chain of rules.

Definition 4: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are rules. $T_1 \rightarrow R_1 \rightarrow C_1$ and $T_2 \rightarrow R_2 \rightarrow C_2$ (T_i triggers rule R_i that commands action C_i). If ($(C_1 == T_2)$ AND $(C_2 == T_1)$) OR ($(T_2 \subseteq C_1)$ AND $(T_1 \subseteq C_2)$), then the rules T_1 and T_2 are triggering each other in a loop resulting in direct looping rule conflict.

Definition 5: $R_1 = \{T_1, C_1, S_1, I_1\}$, $R_2 = \{T_2, C_2, S_2, I_2\}, \ldots, R_n = \{T_n, C_n, S_n, I_n\}$ are chaining rules. Like $T_1 \rightarrow R_1 \rightarrow C_1$ and $T_n \rightarrow R_n \rightarrow C_n$ (T_i triggers rule R_i that commands action C_i). $R_1 \rightarrow R_2 \rightarrow R_3, \ldots, R_n - 1, \rightarrow R_n$ (R_1 triggers R_2, R_2 triggers R_3, and $R_n - 1$ triggers R_n forming a chain of rules). If $C_n == T_1$ OR $T_1 \subseteq C_n$. Then, the R_1 and R_n interaction is an indirect looping conflict.

E. CHAINING RULES

Chaining occurs when the action of one rule is a trigger event for another rule; it can be between two rules, or this relation results in connecting n number of rules. These events may be the states of the actuator or environmental parameters value after executing the rule. We termed partial chaining when the chaining relation is based on either the states of the actuators or the environmental parameters' value. The chaining of rules relation is one of the factors responsible for the interference of the automation rules and results in the complex interaction between the rules. Pure chaining is defined as when one rule depends on another rule to get executed first, like R1, which is only triggered when R2 performs its action.

Definition 6: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are simultaneously executing rules. $T_1 \rightarrow R_1 \rightarrow C_1$ and $T_2 \rightarrow R_2 \rightarrow C_3$ (T_i triggers rule R_i that commands action C_i). If $C_1 == T_2$ OR $T_2 \subseteq C_1$, then R_1 triggers R_2, i.e., R_2 is a chained rule of R_1.

F. CROSS-ENVIRONMENT IMPACT CONFLICT

When different rules command action that affects the same environment parameters in a contrasting manner or the action implications are in contrast to each other, in the same space and time duration. For example, R1 commands open the heater, and R2 commands open the air conditioner simultaneously in the same room or environment, then the heater increases the temperature of the space, and the air conditioner decreases the temperature of the same space. This conflict results in discomfort to the users and more power consumption as the target of both rules will not be achieved and satisfied. This is like the cross-execution conflict where environment parameters are the responsible factor instead of actuator states.

Definition 7: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are simultaneously executing rules. $R_1 \rightarrow I_1$ and $R_2 \rightarrow I_2$ (I_i Impact is the resulting environment state after executing the rule R_i). If $I_1 = \neg I_2$ (where I_1 and $I_2$are opposite impacts), then the rules R_1 and R_2 will result in cross-environmental impact conflict.

Same environment impact conflict occurs when different rules and actions impose the same effect on the same environmental parameter in a smart space. For example, R1 commands open the window, and R2 commands open the air conditioner. Here, both rules aim to lower the temperature, while execution of one among them can achieve the target alone, which will result in a redundant effect on the environment. This conflict is like the rule redundancy where the environment parameter is responsible. Such conflicts result in more power consumption and economic burdens on users.

Definition 8: $R_1 = \{T_1, C_1, S_1, I_1\}$ and $R_2 = \{T_2, C_2, S_2, I_2\}$ are simultaneously executing rules. $R_1 \rightarrow I_1$ and $R_2 \rightarrow I_2$ (I_i Impact is the resulting environmental state after executing the rule R_i). If $I_1 = I_2$, then the rules R_1 and R_2 will result in the same environment impact conflict.

G. CHAINING RESULTED IN RELATION CONFLICTS DETECTION

For detecting the conflicts, considering chaining as the conflict is insufficient for properly detecting the conflicts. In addition, the identification of chained rules and their interference relation with one another are need to be identified. Chaining involves covert triggering, a rule chain may involve two or as many rules as defined by the user. One rule may initiate multiple rule chains; therefore, conflicts detection must also be checked between all the chained rules. In chaining, When one rule triggers another rule then their execution becomes parallel. For example, If R1 triggers R2, then whenever R1 executes, it will also execute R2. Therefore, chained rules execute simultaneously, whether or not their execution times or events overlap. This makes the rule interaction more complex and complicated, which needs to be considered for conflict detection. For example, R1 triggers R2, R2 triggers R3, R3 triggers R4. If R1 commands open the window, R3 commands close the window, and R1 and R3 execution time/event do not overlap. Cross-execution conflict between R1 and R3 will not be detected, if chained rules interrelation is not considered.

H. CHAINED RELATION WITH SIMULTANEOUS EXECUTION

If rule A and rule P are simultaneous executing rules. Assume rule A initiates a chain of rules like $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow F$ and similarly, P initiates $P \rightarrow Q \rightarrow R \rightarrow S \rightarrow T \rightarrow U$ then A and P will be simultaneous with all its chained rules. But what will be the relation of the rule A chained rules (B, C, D, E, F) with P and similarly P chained rules with A. Rule A and rule P execute each other simultaneously. All the chained rules of A will be simultaneously executing rules with P, Q, R, S, and T. P will be with B, C, D, E, and F. Because simultaneous executing rules relation and chaining of rules are transitive in nature, in addition, these are initiated due to the same triggering event directly or indirectly. These add-ons add more complexity to already complex rule interactions. Therefore, in detection, such relations also need to be examined.

I. SAFETY POLICIES

The safety policy is the user's expectation of the security state that should be enforced physically [20]. Safety policies help to identify those rules that, directly or indirectly, through interaction with other rules, lead to the unsafe states of the smart home. Therefore, introducing safety policies in conflict detection provides an extra guard in the conflict detection mechanism.

In the above-discussed literature, existing classification mainly adopted rule redundancy, cross-execution, and cross-environment impact conflict. Only a few include chaining, indirect looping, and the same environment impact conflicts, which are equally important to define and, hence, for detection. Most defined categories in existing

TABLE 1. Existing classification categories related to our defined categories.

Proposed Conflicts	Referenced conflicts (existing)
Rule redundancy	Action Duplicate [15], Shadow Conflict [5], redundancy rules [12], Redundancy conflict [14]
Cross execution	Actuator Race (AR) [21], Action Conflict [15], Execution Conflict [5], independent Conflict [5]. Function-function conflict [9], Functional Conflict [2], Contradiction rule [12]
Rules loop	Loop Triggering [[21], Action Loop [15]
Chaining rules	Transitive-environment-impact-conflict [9], Direct Service Impact Conflict [2], Read write conflict [12], Self-Disabling [21], covert triggering [21], Enabling-Condition Interference and Disabling-Condition Interference [21]
Cross environment impact	Goal Conflict (GC) [21], Action Revert [15], Opposite-environment-impact-conflict [9], Indirect Service Impact Conflict [2]
Same environment impact	Cumulative-environment-impact conflict [9]
Safety Policies	Condition bypass [15]
Relation in detection process	
Chained rules interaction are considered	
Relation of chained rules of one rule with another simultaneously executing rule and its chained rules are considered	None of the prior work, considers these relations in the detection process

classification are subclasses of one or possible cases of their effect. Chaki et al. [2] define four different categories, but these can be detected with the detection of chained rules, and these categories are all possible effects of chaining. Meanwhile, independent conflict and execution conflict [5]; are part of the cross-execution conflict, see Table 1.

Previous work considered different types of classes of rule conflicts but still failed to cover all possible rule adverse interactions, which requires a thorough and complete classification. In the proposed work, we encompass all the existing conflicts and include the same environment impact similar to cumulative-environment-impact conflict [9]. In prior work, only [9] considered such conflict, see Table 2. The same environment impact conflict does not affect the automation or discomfort the users but consumes more power. Such conflict, in some cases, may lead to cross-environment impact. For example, Rule R1 states to open a window, and rule R2 dictates to open the air conditioner, simultaneously aiming to lower the temperature in a room. But if the weather is hot outside, then the temperature of room may increase due to the rule R2 and will reflect cross environment impact. Therefore, detecting Same environment impact conflict is essential for economic power consumption and ecofriendly automation.

On the other hand, the existing detection approaches are only concentrated on finding the discussed conflicts. Still, in support of this, we also require safety policies that check whether the smart home reaches an unsafe/compromised state, and if yes, then Why so? Not only conflicting rules but some non-conflicting benign rules may lead the system to a dangerous state. Therefore, in our proposed ontology, we incorporated safety policies and identified the rules that violate the safety policies directly or indirectly (chained rules) to enhance security.

TABLE 2. Comparison of classification and detection proposed by existing work.

Ref.	RR	CR	LR	CH	CC	CSR	CI	SI	SP
[12]	●	●	○	◑	○	○	●	○	○
[21]	○	●	◑	●	●	○	●	○	○
[4]	●	●	◑	○	○	○	●	○	○
[22]	●	●	○	○	○	○	●	○	○
[13]	●	●	●	○	○	○	●	○	○
[14]	●	○	○	○	○	○	○	○	○
[15]	●	●	◑	○	○	○	●	○	○
[5]	●	●	○	○	○	○	○	○	○
[9]	○	●	○	◑	○	○	●	●	○
[2]	○	●	○	◑	○	○	●	○	○

●denotes existing classification completely identifies the conflict category.
◑denotes existing classification identifies some instances of the conflict category.
○shows classification do not covers conflict category.
(RR) Rule redundancy, **(CR)** Cross execution, **(LR)** Looping rules, **(CH)** Chaining, **(CC)** Chaining resulted relation conflicts, **(CSR)** Chained relation with simultaneous execution considered, **(CI)** Cross environment impact, **SI** Same environment impact and **(SP)** Safety Policies

IV. BACKGROUND

Understanding the smart home and ontology is essential for understanding the proposed ontology mechanism. In this section, we define smart homes and ontology precisely.

A. SMART HOME

Smart homes are the application of the Internet of Things. A smart home enables users to form a smart space that operates on their defined rules and controls it locally and remotely [3]. It comprises of numerous actuators and sensors that change the smart space accordingly [2], [6]. Savvy users may start the AC before reaching home, switch on the coffee maker and start the television to record a program on its arrival or at a particular time. The smart home users install different types of devices and apps. Apps are either mobile apps or dedicated firmware consoles, that provide the interface to the users to define the automation rules. These automation rules connect different devices to accomplish a particular task or achieve a certain comfort level of environmental entities like temperature and humidity [22]. This results in the automated smart environment called a smart home to perform defined activities and tasks. Automating smart homes and their services to users, which vary from utility to security, encourages mass adoption. This has increased demand in recent years and promotes advancement in this domain.

B. ONTOLOGY

Ontology helps to understand the nature of things according to their relationships with others [23], representing the domain [24]. Formal knowledge representation of any system aids in realizing the system before its implementation to diagnose the operational workflow, outcomes, and challenges. This supports developers to avoid flaws and shortcomings in the system [23]. Ontology-based knowledge representation supports explicit conceptualization of the domain, i.e., it defines attributes, entities, and their relationship in a structured manner [25]. Ontology languages like OWL promote ease of expressiveness to represent complex hierarchies, properties, complex relationships, and definitions [26], [27]. Further, ontology is platform-independent and scalable. Promotes reusability, provides the capability of inferencing and reasoning to deduce logical interpretations based on a defined system [28], [29]. Ontology is formed by RDF (Resource description framework), RDFS (RDF Schema) [30] and OWL family of languages [31] that delivers logical knowledge representation [30]. We adopt an ontological way for our proposed work due to the following reasons:

1. *Explicit structure defining:* The smart home automation-related components and their relations are expressed logically and effectively.
2. *Expressivity:* Representation of logical relationships of different entities helps to express the automation rules and their interaction conflicts.
3. *Reasoning and inferencing:* The capability of reasoning helps to identify the direct and indirect interactions of rules and their resulting conflicts.

Based on these points, an ontology is proposed in this paper for the classification and detection of the rule conflicts due to the direct and indirect rule interactions.

V. PROPOSED SMART HOME ONTOLOGY

In this paper, we formed an ontology for detecting the conflicts of the smart home automation rules by representing its automation-related components and formalizing their relationship. We developed our knowledge base in protege [39], [40] and formed SWRL [17] rules to detect the rule conflicts. The proposed ontology may aid the designers in developing smart home frameworks more efficiently to tackle automation rule conflicts at the design phase [4].

A. ONTOLOGY CLASSES

In the smart home, there are various physical and logical components like actuators, sensors, controlling apps, hubs, and cloud for database and computation. In the proposed ontology, we focussed on the components responsible for the automation rules execution, primarily the automation rules and their execution-responsible factors i.e., how and where they interact. Classes of ontology are the conceptual abstract representation of the entities and components. The classes of our ontology represent apps, devices and sensors, permissions, general properties, and safety policies. Among them,

FIGURE 1. Class hierarchy of ontology.

general properties have subclasses like AirFlow_Factors, Heating_Factors and HomeStatus, a combination of the environment entities and the overall status of the smart home (like HomeStatus), as shown in Figure 1.

General properties represent the state of the smart home and the apparent logical interpretation of the device's behavior; for example, Light_Intensity_Factors represent the behavior of devices that affect light intensity. Instances like curtains open in the day and lights ON affect the environment similarly. This is obvious to understand, but in terms of the system, such situations need to be translated into a logic that can be interpreted by the system. Therefore, a general properties class is formed in our ontology for such conditions. Smart home system overall states are not considered in similar prior work. Developing an efficient knowledge base that accommodates parameters that resemble a realistic environment is essential.

Rules are defined through a particular app designated to control smart devices, i.e., automation rules are the sub-class of the Apps class. A rule consists of two major parts: trigger (when it needs to execute) and action (what it needs to do). Triggers and action are the subclasses of the rule's classes. Meanwhile, the permission granted to the apps at installation

TABLE 3. Object properties and relation of the components.

Object property	Relationship among components
App_granted_permission	Domain ∈ Apps, Range ∈ permissions
has	Domain ∈ Apps, Range ∈ rules
hasState	Domain ∈ devices_&_sensors, Range ∈ permission
rule_commands	Domain ∈ permissions, Range ∈ rules
triggers	Domain ∈ rules, Range ∈ permissions
InverseOf	Domain ∈ rules, Range ∈ rules
triggeredBy	Domain ∈ permission, Range ∈ rules
SimulataneousExecution	Domain ∈ rules
device_permission	Domain ∈ devices & sensors, Range ∈ permissions
CrossExecutionConflict	Domain ∈ rules, SafetyPolicies
CrossImpact	Domain ∈ rules
SameImpact	Domain ∈ rules
EnvironmentCrossConflict	Domain ∈ rules
EnvironmentRedundancyConflict	Domain ∈ rules
chaining_app	Domain ∈ rules
IndirectLoopRules	Domain ∈ rules
LoopRules	Domain ∈ rules
RuleRedundancy	Domain ∈ rules
SafetyPolicyViolated	Domain ∈ rules, SafetyPolicies

time represents the class permission. Permission consists of the device's functions invoked according to the rules and during its execution. The rule's components, triggers, and actions are invoked functions or capabilities of the devices & sensors that are granted permission to the app and rules. Therefore, the permission class consists of subclasses of each device, and they contain their functions or permissions as instances, as shown in Figure 1. The SafetyPolicies class has individuals representing safety policies that are checked to identify interactions that lead the smart home to an unsafe state. The devices_&_sensors class represents the devices and sensors installed in the smart home. These are the main components of the smart home related to automation that are considered in our ontology to identify rule conflicts.

B. RELATIONSHIPS AMONG COMPONENTS

Different classes are formed to develop the smart home environment realistically, and their relationship is defined accordingly. For automation, first, the apps are installed in the smart home, and during installation, users grant permission to access the app. These permissions are the capabilities of the actuators and sensors, which the authorized app can access. **App_granted_permission** object property defines the granted permissions by the app. Once the apps are installed, users represent and declare the automation rules in each app for smart home automation. Object property **has** shows the number of the automation rules defined in a particular app. To represent the capabilities of the devices and sensors, the object property **hasState** declared. For each capability of the devices and sensors, corresponding permission is asked by the app, which binds the granted capabilities with the apps to execute the automation. The **device_permission** object property defines such permissions.

Two object properties represent the automation rule, **triggeredBy** and **rule_commands**, before declaring the triggered events and connecting the action commands later. Rules execution changes the states of the actuators and the environmental parameters. Rule execution resulting events may trigger other automation rules; **triggers** object property represent this relation. **triggers** and **triggeredBy** are inverse relations; one connects permissions (granted capabilities) and triggering parameters to the rules. Second connects rules to the triggering events.

To identify the relation between the interaction of the automation rules, different object properties are defined to represent the relation in Table 3. To detect the interaction among the rules, simultaneous executing rules are required to be identified first. As defined above, simultaneous executing rules are due to the same triggering events and overlapping of the triggering parameter duration, such as time and humidity. These simultaneously executing rules are represented by the **SimulatneousExecution** object property, which covers all the pairs of rules triggering simultaneously. Further, for defining the environmental parameters, range data properties like **lowerTime** and **upperTime** for the time, **LowerHumidityRange** & **UpperHumidityRange** for the humidity, **HighRange** & **LowRange** for light intensity, and **LowTempRange** & **UpperTempRange** for temperature are defined. These properties help to find the overlapping parameter ranges of the rules for identifying the simultaneously executing rules.

In addition, object properties like **IndirectLoopRules**, **LoopRules**, and **RuleRedundancy** are defined to represent and collect the rules interaction conflicts. Similarly, for collecting rules that violate safety policies, **SafetyPolicyViolated** object property is defined. Object and data properties defined in Table 3. Which represent the logical connections between the different classes and their instances to enable the developed ontology to retrieve and reason the information from the created knowledge base.

C. SMART HOME RULES

Automation rules are defined by the users, responsible for managing and controlling the smart homes according to user comfort [2], [7]. These rules comprise two main components: triggers and actions [32]. Triggers denote the triggering event when the action must be taken. This triggering event may be a time value, state of the actuator, or environment parameter value range. Meanwhile, the actions define the commands, like where and which command will be executed in a smart environment. Commands and triggering events are the permissions granted to the apps to control actuators and sensors. Rules declared in an app can only bind with the permissions granted to the app. The rule is initiated when its triggering event occurs and terminates when the action is executed. This results in a state change of the smart

home environment [33], [34]. A rule may have multiple triggering events and commands to execute [35]. Once defined, the rules are stored in a database with an identity referenced to the user who created them in a multi-resident smart home, while rules are stored in a user smart home.

When a rule is created, users define the access of the devices and environmental parameters or states by allowing related permission to the App by default to all its rules [36]. In isolation, when automation rules are executed in the smart home, they change the state of the environment parameters and actuators as defined in the action command. However, the smart home contains numerous rules connected over the same set of sensors and actuators. So, two or more rules may request to change the state of the same actuator(s) or may be triggered by sensing or occurrence of the same event. In this regard, rules are interconnected and may interact with one another. Such interrelation only occurs when the rules are simultaneously executed.

The simultaneous executions of rules can have interaction relations due to their action commands' impact on each other. Either directly through the actuator state or indirectly via environmental parameters (write-write relation). Another possibility of the rule's interaction is due to the write-read relation, this relation results in rule chaining. Rule chaining makes the rule interaction more complex as chained rules are simultaneously executing rules, as discussed above. These interactions may adversely affect the environment, actuators function, overall automation and economy. These adverse effect interactions called rule conflicts may be conclusively classified into five categories, as shown in Table 1. Rule conflicts can be defined as write-write and write-read conflicts. Write-write conflicts occur when rules have the same or opposite effect on the actuator state or environmental parameters, leading to four possible interactions. Meanwhile, write-read conflicts arise when one rule changes the state of an actuator or the environmental parameter after its execution, and such a change is a triggering event for another rule that reads it. These results trigger another rule. Further, such an execution-initiation link will create a chain of rules that may involve two or multiple rules. A rules chain involving distinct rules will form a linear chain. A circular chain is formed if the starting and ending rules are the same. In addition, a chain may have linear and circular patterns. Although detecting such rule conflicts is complex to trace in any interaction, it is essential for the efficiency of the smart home, security, users' comfort, and economy.

In the proposed work, we selected rules from existing works and synthesized more complex rules to verify the detection mechanism of our ontology. A set of rules is formed for each rule conflict to ensure that our ontology detects all types of conflicts defined in classification. Some complex rules (multiple triggering and multiple actions) are also formed to further ensure the validity of the proposed work, see Table 6.

TABLE 4. Automation rules formed for evaluation.

Rules	Description of the rules
R1	When the light intensity is below 400 Lux, then open the curtains.
R2	When the light intensity is below 500 Lux, then open the curtains.
R3	When the humidity is below 45% RH, then open the humidifier.
R4	When the humidity is below 65% RH, then close the humidifier.
R5	When the temp is below 24 °C then, open the heater and keep the temp above 26 °C.
R6	When the temp is above 28 °C then, open the air conditioner and keep the temp below 28 °C.
R7	When the air conditioner is open, close the window.
R8	When the window is closed, open the air conditioner.
R9	When the air conditioner is closed, open the window.
R10	When the window is open, close the air conditioner.
R11	When the air purifier is on, then close the window.
R12	When the window is closed, then close the curtains.
R13	When the curtains are closed, then open the air purifier.
R14	When the humidity is above 50% RH, then open the window.
R15	When the humidity is above 60 % RH, open the air purifier.
R16	When the light intensity is below 350 Lux, then open the lights.
R17	When the light intensity is below 500 Lux, then open the curtain.
R18	During the time interval from 8:00 am to 8:00 pm, open the windows and set the home status HOME.
R19	During the time interval 09:00 pm to 07:00 am, close the windows.
R20	When humidity is above 50% RH, then open the windows.
R21	When the motion is detected, then switch off the security camera.
R22	When the home status is HOME, then switch off the security camera.
R23	During the time interval 8:00 am to 11:00 am, close the air conditioner.
R24	When the temperature is above 25°C, then open the air conditioner.
R25	When the smoke is detected, then switch on the fire sprinklers.
R26	When the moisture is detected, turn off the water valve.
R27	During the daytime, switch off the lights and open the curtains.
R28	When the lights are switched on, set the home status to HOME.
R29	When the home status is HOME, then switch on the lights and switch off the security camera.
R30	When the motion is not detected, then set the home status AWAY.
R31	When the home status is AWAY, switch off the lights and the security camera.
R32	At 07:00 pm, switch on the television.
R33	When the television is switched on, then switch on the coffee machine.
R34	When the coffee machine is switched on, then switch on the lights.
R35	When the lights are switched on, then switch on the television.

1) AUTOMATION RULES DEFINED IN OWL

Smart home automation rules, as described above, are composed of triggers and action commands. We represented these automation rules in rudimentary form in our ontology. We defined these rules' triggers and action commands by object properties **triggerredBy** and **rule_commands**. **triggerredBy** object property defines the triggering condition of the automation rule. While **rule_commands** property

assigns a set of action commands for the automation rule to be executed. These properties can represent one to many relations, thus describing the multiple triggering conditions and action commands of each automation rule resembling the realistic definition of such rules.

Triggering conditions for each automation rule are pre-conditions that must be satisfied to execute the rules. Such triggering conditions may be the status of the actuator, temperature, humidity, light intensity range, and particular time or duration. We defined data properties for declaring the triggering parameters of the automation rules. These data properties are **LowerHumidityRange**, **HighHumidityRange** for expressing the humidity range; **HighRange**, **LowRange** for light intensity, **UpperTempRange**, **LowTempRange** for temperature and **lowerTime**, **upperTime** for the time duration. We defined the ranges for the parameters because even a particular value for the triggering parameters can be declared by specifying the same lower and upper bound of the parameter (**lowerTime** = 09:00:00 && **upperTime** = 09:00:00), but vice versa is not possible. The status of the actuator is declared by the **triggeredBy** object property as the triggering parameter for the automation rule, for example rule R1 **triggeredBy** Window_Close actuator state. The actions of the automation rules are the state of the actuators or the environment parameters to be achieved after executing the automation rules. Even the environment parameters preferences are achieved by commanding the actuator's state change. In our ontology, actions of the automation rules are declared by the **rule_commands** object property, for example rule R1 **rule_commands** Window_Close state of the actuator.

Following are the challenges that we faced in defining automation rules in a realistic manner for the ontology:

a: TIME DURATION COMPRISING THE CONSECUTIVE DAYS' TIME IN TRIGGERING PARAMETER:

Automation rules triggered based on time duration may consist of any period of a day or successive days. Suppose rule R3 states close the doors during the night duration, i.e., 10:00 pm to 06:00 am; this type of rule definition is evident and natural in smart home automation. However, defining such rules in the ontology is challenging as time duration cannot be expressed directly. Time format supports 24-hour duration like 00:00:00 to 23:59:59; in this format, duration for consecutive days cannot be declared. Further, the triggered time duration of the rules must be compared to identify simultaneously executed rules; this becomes difficult for defined R3 types of rules. Therefore, to tackle such a hurdle in defining the rules, we split the duration of consecutive days and defined the rules accordingly. For defining R3 in our ontology, we define R3 and R3ext, such as rule R3 states close doors during 22:00:00 to 23:59:59 and rule R3ext dictates close doors during 00:00:00 to 06:00:00. Then we connect R3 and R3ext by **chaining_app** object property (R1 **chaining_app** R2 defines R1 triggers R2); as prior duration triggers next duration. This helps to define consecutive days' time duration in rules, effectively identifies the simultaneously executed rules with R3.

b: MULTIPLE TRIGGERING CONDITIONS

The automation rules are of two types according to the triggering conditions it defines, i.e., complex and simple. If an automation rule consists of one triggering condition, then it is a simple rule, and if it contains more than one triggering condition, then it is a complex automation rule. Suppose automaton rules, rule R1 states when the temp> 27°C and home status is home, then open the window. Rule R2 dictates when the temp> 26°C, then open the air-conditioner, R1 is complex, and R2 is a simple automation rule. Defining the complex automation rule in ontology is difficult by using the object property directly for defining the triggering condition, in our case by **triggeredBy** object property. In ontology, the reasoner compares the instances associated with the properties defined according to their relation. Execution of the rule R1 is only considered when both triggering conditions are satisfied. Still, suppose both the triggering conditions are defined directly in the automation rule. In that case, the reasoner will consider the automation rule R1 as initiated or triggered on the satisfaction of any triggering condition. According to our conflict definition, R1 and R2 were found to be simultaneously executing and cross-environment impact conflicting rules. This will detect the wrong relation and conflict between the R1 and R2 rules.

Therefore, in our ontology, we divided the complex automation rules into simple ones and connected them while defining the rules to avoid wrong detections. Rule R1 is defined and divided into R1 and R1ext. Rule R1 dictates when the home status is home, then open the window. Rule R1ext states when temp> 27°C, open the window. If both the rules are simultaneously executing and conflicting with the R2, then only the conflict of R1 and R2 is considered. This helps to avoid inconsistent and wrong results in the proposed ontology.

c: DEFINING THE LOGICAL RELATION BETWEEN THE INSTANCES IN THE SYSTEM

In a smart home, many realistic apparent ties are hard to describe and represent in the ontology, for example if the water valve is closed, the fire sprinkler will not shower water. This interaction is straightforward example which cannot be directly represented in the ontology and needs to be described and defined. We formed a **General_Properties** class to define these apparent interactions and relations. We described such relations as the rules in the form of triggers and actions. The property for the above mentioned example is P1, that states when the water valve is closed, the sprinkler is closed, i.e., **triggeredBy** watervalve_close actuator state and **rule_commands** sprinkler_close state of the sprinkler. Further, considering them during the detection of rule conflicts will ensure that all the rule interactions are covered, which supports the efficiency and effectiveness of the detection mechanism.

D. INTERACTION OF RULES

To detect the interactions of the automation rules of the smart home, first, we need to identify the rules that are simultaneously executing. Only rules that are simultaneously executed interact with each other directly or indirectly. In our ontology, to detect the simultaneously executing rules, we need to identify the simultaneously triggered rules. Along with it, chained rules of triggered rules also need to be recognized as they are simultaneous with the triggered ones. We checked the triggering parameters of each rule, such as humidity, temperature, time duration, light intensity, and state of the actuator. We compared each rule-triggering value range of the environmental parameter with another rule pairwise; to check whether the rule's defined ranges intersect. If one rule-defined value range intersects with another rule, then they will execute simultaneously in the intersecting value range. Meanwhile, the rules with the same triggering actuator state simultaneously execute rules.

Further, chaining rules of each simultaneously executing rule pair are identified and labeled simultaneous executing rules with the pair. Based on the above discussion, we defined SWRL rules in Table 4 to determine the rules that are simultaneously executing. Once rules that are simultaneously executing are identified, rule interaction conflicts are detected among the simultaneously executing rules.

E. DETECTION OF AUTOMATION RULES CONFLICTS

The method of rule conflict detection is illustrated in Figure 2. First, the simultaneously executing rules are identified based on overlapping parameter durations, triggering states, and chained rules. Once the simultaneous executing rules are identified, the conflicts are detected according to the definitions of the conflicts mentioned by the SWRL rules in Table 4. Among every simultaneously executing rule set, each rule is checked pairwise for the conflicts. In this way, the chained rules of simultaneously running rules are also considered pairwise with each other, ensuring that no rule interaction is avoided. For the detection of safety violations, safety violations are defined as the automation rules, and then each simultaneously executing rule is checked with these rules. In this manner, conflicts and safety violations are detected according to the satisfaction of the conditions of the SWRL detection rules.

VI. EVALUATION AND DISCUSSION

As discussed earlier in this paper, the classification of rule conflicts serves as the backbone for detecting conflicts. The adopted and proposed classification of prior studies only cover some of the conflicts. Based on their limitations shown in Table 2, we proposed the smart home rule conflict classification that is efficient and complete. Detection mechanism is proposed and evaluated against the

TABLE 5. SWRL rules for detecting rule conflicts in the proposed ontology.

UpperHumidityRange(?X1, ?H1), UpperHumidityRange(?X2, ?H2), lessThanOrEqual(?H2, ?H1), LowerHumidityRange(?X1, ?L1), LowerHumidityRange(?X2, ?L2), lessThanOrEqual(?L1, ?L2) -> SimultaneousExecution(?X1, ?X2)
UpperHumidityRange(?X1, ?H1), UpperHumidityRange(?X2, ?H2), lessThanOrEqual(?H1, ?H2), LowerHumidityRange(?X1, ?L1), LowerHumidityRange(?X2, ?L2), lessThanOrEqual(?L1, ?L2), greaterThanOrEqual(?H1, ?L2) -> SimultaneousExecution(?X1, ?X2)
rules(?a), rules(?b), rule_commands(?a, ?c), rule_commands(?b, ?d), chaining_app(?a, ?b), SameAs (?c, ?d) -> RuleRedundancy(?a, ?b)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), SimultaneousExecution(?a, ?c), CrossImpact(?b, ?d) -> EnvironmentCrossConflict(?a, ?c)
HighRange(?X1, ?H1), HighRange(?X2, ?H2), lessThanOrEqual(?H1, ?H2), LowRange(?X1, ?L1), LowRange(?X2, ?L2), lessThanOrEqual(?L1, ?L2), greaterThanOrEqual(?H1, ?L2) -> SimultaneousExecution(?X1, ?X2)
rules(?a), rules(?b), triggeredBy(?a, ?c), triggeredBy(?b, ?c) -> SimultaneousExecution(?a, ?b)
lowerTime(?x1, ?l1), lowerTime(?x2, ?l2), upperTime(?x1, ?u1), upperTime(?x2, ?u2), lessThanOrEqual(?l1, ?l2), lessThanOrEqual(?u2, ?u1) -> consists_time_of(?x1, ?x2)
SimultaneousExecution(?a, ?b), chaining_app(?a, ?c) -> SimultaneousExecution(?b, ?c)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), chaining_app(?a, ?c), CrossImpact(?b, ?d) -> EnvironmentCrossConflict(?a, ?c)
UpperTempRange(?X1, ?H1), UpperTempRange(?X2, ?H2), lessThanOrEqual(?H1, ?H2), LowTempRange(?X1, ?L1), LowTempRange(?X2, ?L2), lessThanOrEqual(?L1, ?L2), greaterThanOrEqual(?H1, ?L2) -> SimultaneousExecution(?X1, ?X2)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), InverseOf(?b, ?d), chaining_app(?a, ?c) -> CrossExecutionConflict(?a, ?c)
lowerTime(?x1, ?l1), lowerTime(?x2, ?l2), upperTime(?x1, ?u1), upperTime(?x2, ?u2), lessThanOrEqual(?l1, ?l2), lessThanOrEqual(?u1, ?u2), greaterThanOrEqual(?u1, ?l2) → consists_time_of(?x1, ?x2)
SafetyPolicies(?a), rules(?b), triggeredBy(?a, ?c), triggeredBy(?b, ?c) -> SimultaneousExecution(?a, ?b)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), SameImpact(?b, ?d), chaining_app(?a, ?c) -> EnvironmentRedundancyConflicts(?a, ?c)
rules(?a), rules(?b), rules(?c), SimultaneousExecution(?a, ?b), SimultaneousExecution(?b, ?c) -> SimultaneousExecution(?a, ?c)
rules(?a), rules(?b), rule_commands(?a, ?c), rule_commands(?b, ?d), SimultaneousExecution(?a, ?b), SameAs(?c, ?d) -> RuleRedundancy(?a, ?b)
HighRange(?X1, ?H1), HighRange(?X2, ?H2), lessThanOrEqual(?H1, ?H2), LowRange(?X1, ?L1), LowRange(?X2, ?L2), greaterThanOrEqual(?L1, ?L2) -> SimultaneousExecution(?X1, ?X2)
consists_time_of(?x1, ?x2) -> SimultaneousExecution(?x1, ?x2)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), SameImpact(?b, ?d), SimultaneousExecution(?a, ?c) -> EnvironmentRedundancyConflicts(?a, ?c)
rules(?x), General_Properties(?z), rule_commands(?x, ?y), triggeredBy(?z, ?y) -> chaining_app(?x, ?z)
rules(?z), General_Properties(?x), rule_commands(?x, ?y), triggeredBy(?z, ?y) -> chaining_app(?x, ?z)
rules(?z), rules(?x), rule_commands(?x, ?y), triggeredBy(?z, ?y) -> chaining_app(?x, ?z)
UpperTempRange(?X1, ?H1), UpperTempRange(?X2, ?H2), lessThanOrEqual(?H2, ?H1), LowTempRange(?X1, ?L1), LowTempRange(?X2, ?L2), lessThanOrEqual(?L1, ?L2) -> SimultaneousExecution(?X1, ?X2)
rules(?a), rules(?b), chaining_app(?a, ?b), rule_commands(?b, ?c), triggeredBy(?a, ?c) -> IndirectLoopRules(?a, ?b)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), InverseOf(?b, ?d), SimultaneousExecution(?a, ?c) -> CrossExecutionConflict(?a, ?c)
General_Properties(?z), General_Properties(?x), rule_commands(?x, ?y), triggeredBy(?z, ?y) -> chaining_app(?x, ?z)
rules(?a), rules(?c), rule_commands(?a, ?b), rule_commands(?c, ?d), triggeredBy(?a, ?d), triggeredBy(?c, ?b) -> LoopRules(?a, ?c)
rules(?a), rules(?b), SafetyPolicies(?c), triggeredBy(?a, ?x), rule_commands(?b, ?y), triggeredBy(?c, ?z), rule_commands(?c, ?w), SimultaneousExecution(?a, ?b), SameAs (?x, ?z), SameAs (?y, ?w) -> SafetyPolicyViolated(?c, ?a)
rules(?a), rules(?b), SafetyPolicies(?c), rule_commands(?a, ?x), rule_commands(?b, ?y), triggeredBy(?c, ?z), rule_commands(?c, ?w), SimultaneousExecution(?a, ?b), SameAs (?x, ?z), SameAs (?y, ?w) -> SafetyPolicyViolated(?c, ?a)
rules(?a), rules(?b), SafetyPolicies(?c), triggeredBy(?a, ?x), triggeredBy(?b, ?y), triggeredBy(?c, ?z), rule_commands(?c, ?w), SimultaneousExecution(?a, ?b), SameAs (?x, ?z), SameAs (?y, ?w) -> SafetyPolicyViolated(?c, ?a)
rules(?a), rules(?b), SafetyPolicies(?c), triggeredBy(?a, ?x), rule_commands(?b, ?y), triggeredBy(?c, ?z), rule_commands(?c, ?w), SimultaneousExecution(?a, ?b), SameAs (?x, ?z), SameAs (?y, ?w) -> SafetyPolicyViolated(?c, ?b)

TABLE 5. *(Continued.)* **SWRL rules for detecting rule conflicts in the proposed ontology.**

rules(?a), rules(?b), SafetyPolicies(?c), rule_commands(?a, ?x), rule_commands(?b, ?y), triggeredBy(?c, ?z), rule_commands(?c, ?w), SimultaneousExecution(?a, ?b), SameAs (?x, ?z), SameAs (?y, ?w) -> SafetyPolicyViolated(?c, ?b)
rules(?a), rules(?b), SafetyPolicies(?c), triggeredBy(?a, ?x), triggeredBy(?b, ?y), triggeredBy(?c, ?z), rule_commands(?c, ?w), SimultaneousExecution(?a, ?b), SameAs (?x, ?z), SameAs (?y, ?w) -> SafetyPolicyViolated(?c, ?b)

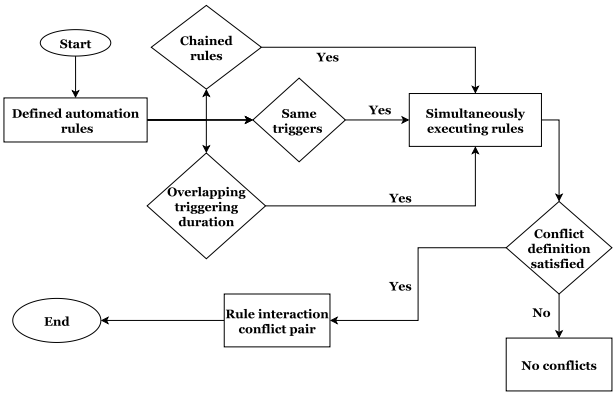

FIGURE 2. Rule conflicts detection process.

TABLE 6. Safety policies defined in smart home.

Safety policies	Description of the safety policy
SP1	The door must be closed when the user is away from home or sleeping.
SP2	When motion is detected then lights must be switched on.
SP3	When the smoke is detected, the door must be unlocked, and the light must be switched on if it is night.
SP4	The light must be switched on when the user arrives home.
SP5	When the user is away, the security camera must be switched on.
SP6	The water valve must be closed when the water sensor is wet and reaches the threshold value defined by the user.
SP7	The windows must be closed when the heater is on.
SP8	When the heater is on, the air conditioner must be off.
SP9	When the air conditioner is on, the heater must be off.

automation rules.To assess the performance of the proposed methodology, we compared it with the existing methods based on the effectiveness and efficiency of identifying the automation rule conflict. Evaluation criteria and results are discussed in the following section. Further, limitations and future work is also highlighted in this section.

A. EVALUATION TEST CASE

To evaluate the proposed mechanism, we formed thirty-five conflicting rules covering all classes of conflicts as the test case. These rules are formed based on the rules defined in [13], [15] and the description of test suite Apps of IoTGuard [20] and Soteria [19]. Further, the safety policies are defined based on IoTGuard, tailored according to the configuration of the defined smart home in the proposed ontology. Rules and safety policies are listed in Table 6 and 7.

B. TEST CASE ANALYSIS

The dataset of rules for the evaluation is developed based on the existing works evaluation test cases and their description of conflicting rules. The dataset comprised, pairs of rules to represent all five classes of conflict. Although our test case dataset is complete to cover all conflict classes, the number of rules is small. While the comparative literature like Soteria [19] evaluated 17 apps comprised 20 violations, IoTGuard [20] evaluated first on 20 apps and second on 65 apps. While, Sun et al., [13] shown their performance result by evaluating six rules as the case study, and Sang et al., [4] selected 32 user policies for the performance analysis. Considering these existing work evaluation dataset, our dataset test cases are sufficient and significant for the evaluation.

C. EFFECTIVENESS

The proposed conflict detection mechanism successfully identifies various types of rule interactions. The number of conflicts detected and identified by the proposed mechanism is shown in Table 5. Interactions like I1 $\Rightarrow$ [(R1, R2), (R1, R27)] (Rule redundancy conflicts), I2 $\Rightarrow$ [(R28, R34)] (Indirect looping), I3 $\Rightarrow$ [(R29, R28)] (Direct looping), I4 $\Rightarrow$ [(R15, R18), (R14, R15)] (Same environment impact), and I5 $\Rightarrow$ [(R18, R29), (R12, R7)] (Chaining conflicts interactions) demonstrate the effectiveness of the proposed mechanism. These interactions are undetectable and unidentified by the existing conflict detection mechanisms due to their incomplete classification and detection coverage, as shown in Table 2. Interactions like I1 are not detected by [2], [9], and [21], and I2 & I3 by [2], [5], [9], [12], [14], and [22] while existing work [4], [15], [21] only detects either I2 or I3. Previous detection mechanisms [2], [4], [5], [12], [13], [14], [15], [21], [22] lack detection of I4-type interactions, and [4], [5], [13], [14], [15], [22] cannot detect I5-type interactions. On the other hand, by considering a comprehensive classification of conflicts, the proposed mechanism significantly enhances the ability to identify complex rule interactions in the smart home environments, filling a critical gap in the existing literature.

D. EFFICIENCY

The efficiency of the proposed conflict detection mechanism is demonstrated through its performance evaluation on the thirty-five conflicting rules, shown in Table 6. The mechanism efficiently detects and identifies conflicts, as evidenced by the number of conflicts detected and identified in Table 5. Our proposed detection mechanism is focused on identifying all implicit interactions to detect all possible conflicts. By focusing on identifying all implicit interactions, including those resulting from the chaining of rules, and simultaneously executing rules relation with each other's chained rules, the proposed mechanism effectively traces complex interactions undetectable by existing mechanisms. Interactions like I6$\Rightarrow$(R28, R34) (Indirect looping conflict) and I7$\Rightarrow$(29,27) (Cross execution conflict) are examples of chained rules interactions with each other and chained rules interactions with another simultaneous executing rule, respectively. In interaction I6, rule R28 triggers the R34 by the changing

TABLE 7. Results of rule conflict detection by proposed method.

Rules conflict	No. of conflicts detected/safety policies violated	% of the occurrence of conflict in n=35 rules
Rule redundancy	63	180
Cross environment impact	7	20
Cross execution	53	151
Same environment impact	36	102
Indirect looping rules	15	42
Direct looping rules	3	8
Chained rules	76	217
Safety violations	4	-
Error detection rate	0 %	
Missed detection rate	0 %	

of rules R28→R29→ R35→R33→R34→R28, which again triggers R28. In interaction I7, R29 is triggered by R18, which executes simultaneously with R27, resulting in an interaction between rule R29 and R27. Detections of I6 and I7 interactions highlight the proposed approach's superior efficiency compared to the existing mechanisms. Additionally, manual tracing of possible conflicting interactions among the rules further validates the correctness and efficiency of the proposed mechanism. Overall, the proposed mechanism not only offers enhanced effectiveness in identifying a wide range of rule interactions but also demonstrates efficiency in detecting conflicts within smart home automation systems.

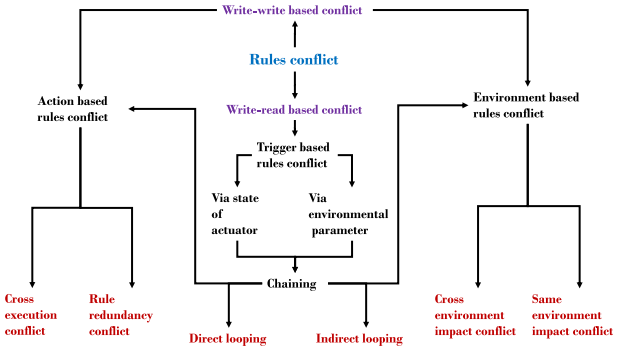

FIGURE 3. Classification of the rule conflicts.

E. CLASSIFICATION OF RULES CONFLICT

In prior work, the classification mainly categorized the cross-execution effect, rule redundancy, and rule looping. Along with these conflicts, few cover cross-environmental impact; only one [9] considered the same environment impact conflict. On the other hand, existing work considered rules chaining as one of the conflicts. But from our point of view, chaining of the rules is not a rule conflict; instead, it may be the reason for the conflicts. Chaining of rules develops covert interactions of the automation rules, which are hard to identify, and conflicts due to them become undetectable.

In existing research, classification categories are interconnected and do not belong or fall into one conflict class. Most existing classifications are the different instances of rule conflicts; execution and indirect conflicts are the instances of cross-execution conflict class [5]. Ideally, classification categories must be mutually exclusive classes to convey a non-ambiguous understanding of the area. In our proposed work, we described the rule conflicts in distinct and mutually exclusive categories, i.e., one conflict class for similar conflicts, encompassing all possible instances of the rule conflict. We defined the conflicts to cover all its rule interactions. In this paper, we categorized the rule conflicts into five categories. Rule redundancy, cross execution, looping rules (direct & indirect rules), cross-environment impact, and same-environment impact conflicts (see Figure 3).

Besides, our conflicts classification categories cover all instances of rule interaction, including complex interactions due to the chaining of rules, which have yet to be considered in prior work. Our classification is unambiguous and complete to support the detection of the rule interaction conflicts effectively and efficiently.

F. RESULTS

The proposed classification and detection mechanism proved to be effective and efficient compared to the existing rule conflict classification and detection mechanisms. The proposed classification, besides covering all the possible instances of conflict class, also considers the same environment impact conflicts, which was not considered in the related works [2], [4], [5], [12], [13], [14], [15], [21], [22] except [9]. The same environment impact conflict and is equally important as other conflicts. Like rule redundancy conflicts, these conflicts may damage the smart devices and prevent the smart home automaton from achieving the user's defined comfort preferences, thus leading to more power consumption and economic loss to the user. Further, the proposed work considered all the implicit interactions efficiently, including the long and complex chaining resulting rules interaction, which are hard to trace and appeared covertly in the defined smart home automation. Besides identifying the rule interactions, it also detects safety policy violations, providing an additional security layer to detect all the conflicting rules. Therefore, it can identify all the possible rule interaction conflicts and shows better results than the existing detection mechanisms.

G. PROPOSED ONTOLOGY FOR MULTI-RESIDENT RULE CONFLICT DETECTION

The proposed ontology efficiently and effectively classifies and detects rule interaction conflicts. The ontology was formed and validated on the single-user smart home environment, which is the limitation of our work. However, our proposed work can be utilized and tailored for the multi-resident smart home environment. Rules of automation in the smart home are declared with the automation app, and one app may define multiple rules with different parameter values. Similarly, in a multi-resident smart home environment, the various users define rules via the same app; in both cases, a complex set of rules is formed. Our

ontology considered each defined rule in the smart home while detecting the conflicts. In this regard, our ontology will detect the rule conflicts in multi-resident smart homes with the same efficiency and effectiveness because rule conflict detection involves a comparison of each rule, either defined by one user or multiple users in the same way. In contrast, single-user and multiuser smart home environments differ when resolving the rule conflicts.

H. LIMITATIONS

Our proposed work effectively and efficiently identifies the rule conflicts, however it does have certain limitations which are as follows.

1) CONTINUOUS STATE CHANGE

One of the limitations of the proposed ontology is that it only depicts an instance of the system, i.e., a static representation of the system [2]. Ontology cannot represent the system's dynamic nature or continuous parameter value change [7], [37]. In the smart home, the automation continuously engages the different parameters of the smart home to enable defined automation. The ontology shows the defined parameter value of the interaction of automation rules and their relation with the other rules and the smart home system. It does not monitor the continuous state change of the smart home automation. However, such limitations do not affect the efficiency of the detection mechanism of the proposed work. Because detection involves defined automation rules involving their parametric static values. Monitoring continuous state change helps us understand the system's flow and track malicious activity or behavior; it is beyond the scope of our work. Secondly, our proposed ontology gives an understanding of the interaction of rules and ways to detect the rules conflicts. This helps to develop a conflict detection mechanism effectively and resist such rule interactions and their adverse effect at the design phase of a smart home system or framework.

2) RESOLUTION OF CONFLICTS

In the prior work, the resolution of the rule conflicts is proposed [2], [5], [7], [21], [38]. Most existing work suggested a priority-based conflict resolution method due to its natural way of resolving the conflict. In the smart home, the users are of different age groups and responsibilities; usually, the decision-making power of each user is different. So, as the priority of the user-defined rules in the smart home. But priority-based conflict resolution may constantly suppress a user's automation in a multi-resident environment whenever the conflict arises. Therefore, we require better mechanisms to resolve rule conflicts. However, we do not propose a conflict resolution scheme in this work.

I. FUTURE DIRECTION

In this paper, we developed a detection mechanism for automation rule conflicts. As discussed above, ontology has limitations like representing the continuous state change and complications in defining the complex rules and realistic properties. In this light, we will extend our work to develop the automated detection mechanism based on the proposed ontology to aid smart home users. Identifying existing rules related to the newly defined rule based on their triggering parameters and actions and alerting the users may help to reduce the number of conflicting rules and safety violations. Adopting such approaches in automated detection tools/methods may enhance the safety and security of the smart home.

VII. CONCLUSION

The conflict between smart home automation rules is an open problem that may leverage attackers to sabotage overall security and harm users. To address such a problem, we proposed an ontology based model to detect smart home rule conflicts. The proposed ontology overcomes the limitations of the existing detection mechanisms by categorizing the conflicts into five distinct classes. In addition, the ontology can detect safety violations to enhance the security of the smart home further. The proposed rule conflicts classification covers all the conflicts and their instances, ensuring completeness and unambiguity to support a robust and efficient detection mechanism. The proposed detection mechanism is evaluated against the thirty-five rules formed based on the existing literature evaluation test cases. The results of our detection method show that the proposed work is effective and efficient in detecting the rule conflicts. Further, an ontological solution will aid the designers in realizing the problem of rule conflicts in the development life cycle due to its scalable, adaptable and shareable nature. In the future, we will enhance the ontology and develop a automated conflict detection tool.

REFERENCES

[1] L. D. Xu, W. He, and S. Li, "Internet of Things in industries: A survey," *IEEE Trans. Ind. Informat.*, vol. 10, no. 4, pp. 2233–2243, Nov. 2014.
[2] D. Chaki, A. Bouguettaya, and S. Mistry, "A conflict detection framework for IoT services in multi-resident smart homes," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Oct. 2020, pp. 224–231.
[3] Q. Wu, G. Ding, Y. Xu, S. Feng, Z. Du, J. Wang, and K. Long, "Cognitive Internet of Things: A new paradigm beyond connection," *IEEE Internet Things J.*, vol. 1, no. 2, pp. 129–143, Apr. 2014, doi: 10.1109/JIOT.2014.2311513.
[4] H. Hu, J. Sang, C. Ye, R. Li, L. Fu, D. Yang, H. Xiang, and C. Fu, "Semantic web-based policy interaction detection method with rules in smart home for detecting interactions among user policies," *IET Commun.*, vol. 5, no. 17, pp. 2451–2460, Nov. 2011, doi: 10.1049/iet-com.2010.0615.
[5] T. Shah, S. Venkatesan, T. Ngo, and K. Neelamegam, "Conflict detection in rule based IoT systems," in *Proc. IEEE 10th Annu. Inf. Technol., Electron. Mobile Commun. Conf. (IEMCON)*, Oct. 2019, pp. 276–284.
[6] B. Huang, A. Bouguettaya, and A. G. Neiat, "Discovering spatio-temporal relationships among IoT services," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Jul. 2018, pp. 347–350.
[7] R. Camacho, P. Carreira, I. Lynce, and S. Resendes, "An ontology-based approach to conflict resolution in home and building automation systems," *Expert Syst. Appl.*, vol. 41, no. 14, pp. 6161–6173, Oct. 2014.
[8] H. Ibrhim, H. Hassan, and E. Nabil, "A conflicts' classification for IoT-based services: A comparative survey," *PeerJ Comput. Sci.*, vol. 7, p. e480, Jan. 2021.

[9] B. Huang, H. Dong, and A. Bouguettaya, "Conflict detection in IoT-based smart homes," in *Proc. IEEE Int. Conf. Web Services (ICWS)*, Sep. 2021, pp. 303–313.

[10] A. Ranganathan and R. H. Campbell, "An infrastructure for context-awareness based on first order logic," *Pers. Ubiquitous Comput.*, vol. 7, no. 6, pp. 353–364, Dec. 2003.

[11] Y. Xu, W. Niu, H. Tang, G. Li, Z. Zhao, and S. Ci, "A policy-based web service redundancy detection in wireless sensor networks," *J. Netw. Syst. Manage.*, vol. 21, no. 3, pp. 384–407, Sep. 2013, doi: 10.1007/s10922-012-9237-1.

[12] Y. Sun, T.-Y. Wu, X. Li, and M. Guizani, "A rule verification system for smart buildings," *IEEE Trans. Emerg. Topics Comput.*, vol. 5, no. 3, pp. 367–379, Jul. 2017, doi: 10.1109/TETC.2016.2531288.

[13] Y. Sun, X. Wang, H. Luo, and X. Li, "Conflict detection scheme based on formal rule model for smart building systems," *IEEE Trans. Hum.-Mach. Syst.*, vol. 45, no. 2, pp. 215–227, Apr. 2015, doi: 10.1109/THMS.2014.2364613.

[14] Z. Lin, T.-Y. Wu, Y. Sun, J. Xu, and M. S. Obaidat, "A TAS-model-based algorithm for rule redundancy detection and scene scheduling in smart home systems," *IEEE Syst. J.*, vol. 12, no. 3, pp. 3018–3029, Sep. 2018, doi: 10.1109/JSYST.2017.2771349.

[15] 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. ACM SIGSAC Conf. Comput. Commun. Secur.*, Nov. 2019, pp. 1439–1453, doi: 10.1145/3319535.3345662.

[16] D. T. Nguyen, C. Song, Z. Qian, S. V. Krishnamurthy, E. J. M. Colbert, and P. McDaniel, "IotSan: Fortifying the safety of IoT systems," in *Proc. 14th Int. Conf. Emerg. Netw. Exp. Technol.*, Dec. 2018, pp. 191–203, doi: 10.1145/3281411.3281440.

[17] I. Horrocks, P. F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, and M. Dean. *SWRL: A Semantic Web Rule Language Combining OWL and RuleML*. Accessed: Jan. 1, 2024. [Online]. Available: https://www.w3.org/submissions/SWRL/

[18] C.-J.-M. Liang, B. F. Karlsson, N. D. Lane, F. Zhao, J. Zhang, Z. Pan, Z. Li, and Y. Yu, "SIFT: Building an Internet of Safe Things," in *Proc. 14th Int. Conf. Inf. Process. Sensor Netw.*, Apr. 2015, pp. 298–309, doi: 10.1145/2737095.2737115.

[19] Z. Berkay Celik, P. McDaniel, and G. Tan, "SOTERIA: Automated IoT safety and security analysis," in *Proc. USENIX Annu. Tech. Conf.*, 2018, pp. 147–158. [Online]. Available: http://www.usenix.org/conference/atc18/presentation/celik

[20] Z. B. Celik, G. Tan, and P. McDaniel, "IoTGuard: Dynamic enforcement of security and safety policy in commodity IoT," in *Proc. Netw. Distrib. Syst. Secur. Symp.* Reston, VA, USA: Internet Society, Mar. 2019, doi: 10.14722/ndss.2019.23326.

[21] H. Chi, Q. Zeng, X. Du, and J. Yu, "Cross-app interference threats in smart homes: Categorization, detection and handling," in *Proc. 50th Annu. IEEE/IFIP Int. Conf. Dependable Syst. Netw. (DSN)*, Jun. 2020, pp. 411–423, doi: 10.1109/DSN48063.2020.00056.

[22] M. Shehata, A. Eberlein, and A. Fapojuwo, "Using semi-formal methods for detecting interactions among smart homes policies," *Sci. Comput. Program.*, vol. 67, nos. 2–3, pp. 125–161, Jul. 2007, doi: 10.1016/j.scico.2006.11.002.

[23] P. Clark. (1996). *Requirements for a Knowledge Representation System*. [Online]. Available: https://allenai.org/content/team/peterc/working-notes/010.pdf

[24] G. Rocher, J.-Y. Tigli, and S. Lavirotte, "Semantic inferences towards smart IoT-based systems actuation conflicts management," in *Proc. IFIP Int. Internet Things Conf.*, 2022, pp. 255–273.

[25] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993, doi: 10.1006/knac.1993.1008.

[26] R. Studer, V. R. Benjamins, and D. Fensel, "Knowledge engineering: Principles and methods," *Data Knowl. Eng.*, vol. 25, nos. 1–2, pp. 161–197, Mar. 1998, doi: 10.1016/s0169-023x(97)00056-6.

[27] D. L. McGuinness and F. van Harmelen. *OWL Web Ontology Language*. W3C. Accessed: Jan. 1, 2024. [Online]. Available: https://www.w3.org/TR/owl-features/

[28] N. Guarino, "Understanding, building and using ontologies," *Int. J. Hum.-Comput. Stud.*, vol. 46, nos. 2–3, pp. 293–310, Feb. 1997.

[29] C. Islam, M. A. Babar, and S. Nepal, "An ontology-driven approach to automating the process of integrating security software systems," in *Proc. IEEE/ACM Int. Conf. Softw. Syst. Processes (ICSSP)*, Nepal, May 2019, pp. 54–63.

[30] B. McBride, "The resource description framework (RDF) and its vocabulary description language RDFS," in *Handbook on Ontologies*. Cham, Switzerland: Springer, 2004, pp. 51–65.

[31] F. Van Harmelen and D. L. McGuinness, "OWL web ontology language overview," *World Wide Web Consort. Recomm.*, vol. 69, p. 70, Jan. 2004.

[32] T. Perumal, M. N. Sulaiman, and C. Y. Leong, "ECA-based interoperability framework for intelligent building," *Autom. Construct.*, vol. 31, pp. 274–280, May 2013.

[33] Y. Yu and J. Liu, "TAPInspector: Safety and liveness verification of concurrent trigger-action IoT systems," *IEEE Trans. Inf. Forensics Securityy*, vol. 17, pp. 3773–3788, 2022.

[34] C.-J.-M. Liang, L. Bu, Z. Li, J. Zhang, S. Han, B. F. Karlsson, D. Zhang, and F. Zhao, "Systematically debugging IoT control system correctness for building automation," in *Proc. 3rd ACM Int. Conf. Syst. Energy-Efficient Built Environ.*, Nov. 2016, pp. 133–142.

[35] V. Zhao, L. Zhang, B. Wang, M. L. Littman, S. Lu, and B. Ur, "Understanding trigger-action programs through novel visualizations of program differences," in *Proc. CHI Conf. Hum. Factors Comput. Syst.*, May 2021, pp. 1–17.

[36] E. Fernandes, J. Jung, and A. Prakash, "Security analysis of emerging smart home applications," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2016, pp. 636–654, doi: 10.1109/SP.2016.44.

[37] T. Bittner, "Formal ontology of space, time, and physical entities in classical mechanics," *Appl. Ontology*, vol. 13, no. 2, pp. 135–179, May 2018.

[38] S. Munir and J. A. Stankovic, "DepSys: Dependency aware integration of cyber-physical systems for smart homes," in *Proc. ACM/IEEE Int. Conf. Cyber-Phys. Syst. (ICCPS)*, Apr. 2014, pp. 127–138, doi: 10.1109/ICCPS.2014.6843717.

[39] R. Sivakumar and P. V. Arivoli, "Ontology visualization PROT tools—A review," *Int. J. Adv. Inf. Technol.*, vol. 1, no. 4, pp. 1–11, Aug. 2011. [Online]. Available: https://api.semanticscholar.org/CorpusID:5379305

[40] J. H. Gennari, M. A. Musen, R. W. Fergerson, W. E. Grosso, M. Crubézy, H. Eriksson, N. F. Noy, and S. W. Tu, "The evolution of Protégé: An environment for knowledge-based systems development," *Int. J. Hum.-Comput. Stud.*, vol. 58, no. 1, pp. 89–123, Jan. 2003, doi: 10.1016/s1071-5819(02)00127-1.

[41] R. Arp, B. Smith, and A. D. Spear, *Building Ontologies With Basic Formal Ontology*. Cambridge, MA, USA: MIT Press, 2015. [Online]. Available: https://mitpress.mit.edu/9780262527811/building-ontologies-with-basic-formal-ontology/

[42] F. Azzedin, M. Eltoweissy, and S. A. Khwaja, "Overview of service oriented architecture for resource management in P2P systems," in *Handbook of Research on P2P and Grid Systems for Service-Oriented Computing*. Hershey, PA, USA: IGI Global, 2010, pp. 175–196, doi: 10.4018/978-1-61520-686-5.ch008.

[43] K. Baclawski and T. Schneider, "The open ontology repository initiative: Requirements and research challenges," in *Proc. Workshop Collaborative Construct., Manag. Linking Structured Knowl. ISWC*, 2009, p. 18.

ADEEB MANSOOR ANSARI received the B.Sc. degree in computer applications and the Master of Computer Applications degree from Aligarh Muslim University. He has been a Researcher with Jamia Millia Islamia, since 2018, his significant contributions span the broader spectrum of *Computer Science*, with a particular focus on fortifying security in smart homes. He is currently pursuing the Ph.D. degree with Jamia Millia Islamia. This reflects his dedication to holistic advancements in technology and knowledge. His research, though centered on this pivotal area, showcases versatility by extending into domains, such as the IoT, CPS, and information technology.

MOHAMMED NAZIR received the master's degree in computer applications from AMU, Aligarh, and the Ph.D. degree from JMI, New Delhi. He is currently a Professor with the Department of Computer Science, Faculty of Science, Jamia Millia Islamia (A Central University), New Delhi, India. He has more than 21 years of teaching experience at PG level. He has supervised three Ph.D. theses in the area of software security. He has participated in several national and international level conferences/workshops. He delivered talks in some of the international/national level conferences/workshops. He has more than 40 research publications in journals and conference proceedings of national and international repute. His research interests include software security and software quality assurance. He is a Life Member of Indian Society for Technical Education (ISTE).

KHURRAM MUSTAFA is an IIT Delhi Alumnus, who is currently a senior-most Professor with the Department of Computer Science, Jamia Millia Islamia (A Central University), New Delhi, India. Despite having completed his Ph.D. on a topic related to eLearning, he continues to supervise students and write/speak on information security, e-learning, and research methods. During his five-year hiatus, he worked as a Professor/Associate Professor at universities in Saudi Arabia, Yemen, and Jordan. He was also the Principal Investigator of a three-year government-funded information security project and delivered more than 60 invited talks, including several keynote addresses. In addition to authoring *Scientific Research Primer* (Ane Books, 2021) and coauthoring two other books, *Software Quality: Concepts and Practices* and *Software Testing: Concepts and Practices* (both published by Narosa, India, and Alpha Science, U.K.), he has mentored more than a dozen Ph.D. candidates. The latter's Chinese edition has also been released. Aside from these, he has coauthored more than a dozen book chapters and more than 100 research articles published in international journals/proceedings. He is also a member of several professional scientific societies, including ISTE, ICST, CSI, EAI, ACM-CSTA, eLearning Guild, and InfoPier, and several academic committees and editorial review boards.