Automating Conflict Detection and Mitigation in Large-Scale IoT Systems

Pavana Pradeep, Amitangshu Pal, Krishna Kant
Computer and Information Sciences, Temple University, Philadelphia, USA
Email: {pavana.pradeep, amitangshu.pal, kkant}@temple.edu

***Abstract*—In this paper we examine the problem of conflict detection and mitigation across multiple independently designed IoT subsystems deployed in a shared environment. The desired behavior of the system is codified in terms of predefined "safety properties". We allow both the operational rules and safety properties to include time and temporal logic operations and detect their potential violation proactively via a "look ahead" mechanism. The problematic operational rules are then perturbed within the allowable range for mitigation. We show that our mitigation approach, based on intelligent combinatorial optimization, can resolve the conflicts via perturbation in 100% of the cases where such a resolution is feasible.**

***Index Terms*—Conflict detection, IoT system, safety properties, combinatorial optimization, linear temporal logic**

I. Introduction

With the recent wave of technological advancements, the deployment of the "Internet of Things" (IoT) is becoming ubiquitous, ranging from common home and personal appliances to sophisticated safety-critical systems such as nuclear plants and medical implants. Maintaining a safe and secure operation of such IoT systems is of prime importance, as the functionality of these systems relies on the entrusted automation. Of the many types of malfunctions that could occur in IoT systems, an important class is of conflicting operation of various actuators, usually caused by misconfiguration of rules. The purpose of this paper is thus to explore mechanisms for detecting and automatically mitigating conflicts that may arise between subsystems that are designed/owned by different parties. While the mechanisms that we develop are general, we will ground them in the specific context of Intelligent Building Management System (IBMS) – a prime area for the application of IoT technologies.

A. Related works and their limitations

The topic of conflict detection and resolution in IoT devices and systems has been considered extensively in the literature. Vannucci et al. [1] proposes a verification framework with satisfiability modulo theory (SMT) for a smart environment with respect to event-condition-action (ECA). Sun, et al., [2] specifies the relation among all building management rules and classifies rule conflicts into five types. These correspond to some detailed deficiencies that may appear in the rules (e.g., two rules specifying the same action under different conditions or various types of loops in actuations). CityGuard [3] proposed an approach to intercept action for smart service to detect and resolve conflicts for smart cities. Ma, et al. [4] from the same group discusses various types of conflicts in smart cities (including device, environment, human, and topology) and their consequences. A study of conflicts in 35 home automation systems in Munir, et al. [5] shows that many conflicts can occur, and the authors devise a runtime conflict checking mechanism called Depsys. It attempts to resolve conflicts after the fact based on simple application priorities. But as discussed in the survey paper [6], most techniques do not consider timing related properties which are our main focus. Also, the conflicts are detected either via a static modeling checking or, more often, dynamically at run time. The conflict resolution is invariably done by defining priorities among rules or devices so that a lower priority rule/device is blocked in case of conflict.

This research was supported by NSF grant CNS-1527346.

B. Our objective

We consider a system of subsystems where each subsystem has its own set of operational rules (ORs) designed not to conflict with one another. However, conflicts can arise across subsystems. The proper operation of the entire system is defined by a set of "safety properties" (SPs) that essentially take the same form as the operational rules. Both the ORs and SPs may involve time or time duration and are expressed in linear temporal logic (LTL). The inability to simultaneously satisfy both SPs and ORs is considered a conflict. Since the safety property is a necessary condition, one possible way to avoid the conflict is by "weakening" the conflicting rule(s). This weakening is possible in most (but not all) cases since the rules in this environment will invariably involve some thresholds (e.g., temperature$>75F$) and time periods (e.g., [10 pm to 5 am] clock time), both or either of which may have some flexibility. The idea is that if these thresholds and/or times can be altered somewhat, conflicts can be avoided rather than having to block one of the conflicting actions. It is assumed that additional rules (or constraints) provide upper and lower bounds on these quantities so that the alteration attempt always keeps them within those bounds. Nevertheless, the altered values may be presented as a suggestion to the user/operator for approval first rather than applying them automatically.

Since IoT systems operate in a dynamic environment with changing conditions, the ability to alter the rule parameters dynamically provides a powerful capability to "tune" them suitably for the prevailing conditions (e.g., cold weather, hot weather, night-time, time of special social events, etc.). Instead of letting conflicts occur and resolving them by blocking one

of the actions, we can make suitable changes to the rules to avoid the conflicts under those conditions. The changes could be either temporary or perhaps made permanent by introducing additional condition-specific rules. In either case, we need the capability to monitor for different situations/conditions. The permanent addition may appear to be a more attractive solution for the future. However, it must be noted that each addition of an exception condition makes the rules more complex and more challenging to handle for the next conflict.

C. Our contributions

Since most events in an IoT system can occur almost anytime, a static conflict resolution faces huge challenges. First, we need to consider the worst-case situations, which are likely to be the ones where most events happen concurrently, including even the rare ones (e.g., a fire). This may indicate difficult-to-resolve conflicts that almost never occur in reality. Second, even the worst-case scenarios may have many possibilities, all of which must be analyzed. On the other hand, dynamic conflict resolution occurs after the conflict has already been detected. Thus the only possible resolution is to either block some action to satisfy the safety property or, if possible, ignore the safety property. For example, if the safety property says that a switch should not be turned on within 1 minute of being turned off, we may occasionally let it happen. Accordingly, the key contributions and novelty of this paper are as follows:

1) A systematic method to consider inter-subsystem interaction by defining a set of safety properties that must be enforced, potentially weakens the operational rules.
2) Modeling IoT conflicts in a dynamic environment where both the operational rules and safety properties involve time and thus the avoidance/resolution could also be time dependent.
3) An automated mechanism to minimally alter the conditions enabling a conflicting action so as to remove the conflict. To the best of our knowledge, this is the first attempt to resolve IoT conflicts via an automatic modification of the rules.

To limit the scope of the paper, we assume that all devices and subsystems are functioning normally. In particular, the conflicts resulting from IoT devices that are unreachable or non-responsive, operating intermittently, providing incorrect data, or intelligently misbehaving due to attacks/compromise are not addressed here. We also do not explicitly address device mobility, although the dynamic conflict avoidance described above applies nicely to mobile environments where the conditions may change dynamically.

D. Paper organization

The rest of the paper is organized as follows. In section II we describe the key conflicts and other issues that arise in large scale IoT systems. Section III then describes a model of the IoT infrastructure in terms of operational rules and safety properties. Section IV presents an approach for resolving the conflicts via minimal cost alteration for the rules. Section V presents a comprehensive evaluation of the proposed algorithms. Finally, Section VI concludes the discussion.

II. Dependencies and Conflicts in IoT Systems

A large scale IoT system will have many sensors and actuators and a plurality of controllers and control loops even for the same physical area. For example, an IBMS may have separate controllers for energy and security management functionality, which have a peer-to-peer relationship. In other instances, a lower-level controller may operate at a more refined time granularity and interact with a higher-level controller. For example, the surveillance controller may continuously track people's movement and activities and interact with the building-access controller, which operates at a coarser time granularity.

The *operational policies* followed by an IoT controller are typically parameterized with the supported features, such as desired temperature setpoint or range. These parameters themselves may be changed manually by the operator or autonomously. A policy can be thought of as the pair (*trigger, action*) where the trigger is a condition or event based on the sensed values and an action is an assertion of control by allowing/blocking actuation of some actuators.

Multiple controllers become interdependent because of potentially overlapping operational policies. This happens for several reasons, as discussed in [7]. Still the main ones are as follows: (a) different controllers depending on wanting to control the same actuator, (b) action of other controllers depending on the same sensors or parameters of the same shared environment.

The dependencies between various *operational policies* can cause *conflicts* with potentially serious consequences with respect to operation and compromise by attackers [7]. However, they can also be beneficial in that they can be used for detecting anomalies and malfunctioning devices (e.g., a temperature sensor showing a very different reading than others in a shared area). In either case, we could define a set of *safety policies*), to ensure that the system does not end up in undesirable states.

IoT deployments continue to evolve in terms of physical aspects (for example, new devices added, old ones upgraded or retired), and the operational environment (for example, new or different vulnerabilities). Therefore a collective static and dynamic analysis of the subsystems and their interactions is essential to understand various forms of conflicts, their consequences, and mitigation. Furthermore, this must be done in an environment where time plays a vital role in the correctness of operation, the incidence of conflicts, and their mitigations.

III. IoT Collaboration and Conflict Model

A. Operational Policies and Rules

An *operational policy* $\mathcal{P}_i$ for controller i essentially moves the subsystem from one state to another. The policy can be expressed as the following triple:

$$\mathcal{P}_i = [C_i^{(pre)}, \mathbb{A}_i, C_i^{(post)}] \quad (1)$$

Where $C_i^{(pre)}$ and $C_i^{(post)}$ are the precondition and postcondition assertions about the state before and after taking the stated action $\mathbb{A}_i$. The index i here emphasizes that this policy deals only with what is visible to controller i. The pre/post conditions are Boolean expressions over various state variables, including the status of an actuator and the sensor's values. In many cases, the Boolean expression is a simple comparison of the state variable, e.g., "door_camera=on & time_of_day<18"; however, it can also involve complex analytics over the captured data as in "unknown_person_seen_by(door_camera)". The evaluation of such functions could challenge real-time operation of the IoT system; however, this paper is not focused on that aspect of the problem.

An operational policy can also be considered as an *operational rule*, denoted $\mathcal{R}_i^{(o)}$, expressed in first order logic by simply viewing it as follows:

$$\mathcal{R}_i^{(o)} = [C_i^{(pre)} \& \mathbb{A}_i \Longrightarrow C_i^{(post)}] \quad (2)$$

As a concrete example, consider a policy that turns on cooling when the room temperature goes above 25C. This can be expressed as [(temp>25 & cooling=off & turnon_cooler) $\Longrightarrow$ cooling=on]. As another example, rule R4 in Table I can be expressed at [Motion & luminance$< B_0$ & turnon_lights $\Longrightarrow \forall_i \text{light}_i = on$].

B. Safety Properties

A safety property expresses the idea that "something (bad) should *not* happen" during the system execution. Two examples of safety property are: *if smoke detected then open windows*, and *if garage-door opened for more than 'n' hours then close the garage-door*. Safety properties provide a generic way of stating what is considered to be "conflict". Since we assume that no conflicts occur within a subsystem (or, instead, any such intra-subsystem conflicts have already been dealt with), the safety properties necessarily go across multiple subsystems.

In general, the rules of two different subsystems may have a *direct conflict* in that they imply opposing actions for the same actuator under a common underlying condition. For example, if two rules ask for opening and closing the door, this is a direct conflict. A direct conflict means that for some action A, we have two rules $C_1 \Longrightarrow A$ and $C_2 \Longrightarrow \not A$ and both conditions C_1 and C_2 can be simultaneously true. The two rules may be untimed (i.e., "always" hold), or hold eventually, during some slot, until some event happens, etc.

In addition to direct conflicts, *Extended conflicts* can be specified through safety properties. For example, opening and closing the window within a few minutes can be considered a conflict and checked via a safety property. Similarly, two actuators' simultaneous operation that affects the same state variable (e.g., heater and cooler) will not be recognized as a conflict unless a safety property prohibits it.

Both types of conflicts generally involve enabling conditions. If these conditions cannot be changed to avoid the conflict, the only way to resolve the conflict is to block one of the conflicting actions, as suggested in much of the prior work on the subject [3]–[5], [8]. In case of safety property (SP) conflicting with a rule, the rule's action should be the one that is blocked (else the concept of SP is meaningless); otherwise, it is necessary to prespecify relative importance or cost of blockage to make the decision.

The more interesting case that we address here is when the enabling condition of a rule involves some thresholds which can be altered to avoid the conflicts. For example, consider the following rules where DO means door-open and DC means door-closed:

HVAC: if [(Temp>80F $\wedge$ DC_time>5 min) $\vee$ (CO2_level>5% $\wedge$ DC_time>7 min)], open_door
Security: if (DO_time>3 min), close_door

and the safety property:

DO_time > 10 min

Here the conflict can be avoided by lengthening the open or close durations. More generally, we try to resolve the conflicts by "weakening" the operation rules.

C. Formal Modeling of Conflicts

The conflict between safety properties (SPs) and operational rules (ORs) is traditionally cast as a Boolean satisfiability problem of first-order logic. Still, with a significant extension: since both SPs and ORs involve at least the comparisons and arithmetic, we need to include the appropriate "theories" to deal with them. This is the domain of satisfiability modulo theories (SMT), which can solve huge problems scalability [9]. Several tools, such as the popular Z3 [10] already have built-in these theories built-in. However, we need two additional capabilities to deal with a predictive model of IoT operations in a cyber-physical system such as the smart home. This includes (a) theories associated with smart-home processes and (b) an explicit involvement of time and temporal concepts. The two are actually related since time is an essential component for tracking these processes.

The nature of theories required for expressing and checking the ORs and SPs depends on the degree of predictability included therein. On one extreme, if the rules are purely driven by actual events (e.g., temperature exceeding 80F). In this case, the theories only need to specify proper semantics of the operations and rule out impossible state transitions and actions (e.g., window-close has an inverse effect of window-open, the window cannot be locked/unlocked if it is open, etc.). On the other extreme, if we want to have predictive rules (e.g., rules accounting for the impact of the window open on the temperature within 10 minutes), the entire heat-transfer physics (or at least a simplified version of it) must be added to the model. We would need some of the latter, as described in the next subsection.

For modeling temporal properties, we use Linear Temporal Logic (LTL), which allows the expression of concepts like eventually, until, as long as, at some time, etc. LTL formulae are built up from a set of atomic propositions (AP), with some operators, the logical operators $\neg$ (negation), $\wedge$ (conjunction), $\vee$ (disjunction) and temporal operators G(always), X(next), U(until), F(eventually). Formally, an LTL formula ϕ can be

defined starting with a simple proposition p and recursively applying the following operators:

$$\phi ::= p \mid \neg\phi \mid \phi \wedge \phi \mid \phi \vee \phi \mid X\phi \mid \phi U \phi \mid G\phi \qquad (3)$$

An LTL formula is satisfiable by a model if there exists a sequence of states in the model such that the formula is true from a given initial state.

However, LTL does not deal with real-time, which is crucial for expressing the operations in an evolving cyber-physical system. We address this by making the time-slotted. One could then speak of next or previous time-slots. For convenience, we define time-slots of a few different durations to represent events at different time scales. With this, SMT itself needs to be extended, and we use the existing NuXMV model checker [11]. The NuXMV model checker uses binary decision diagram (BDDs) based model checking with SAT-based model checking to address the state-space explosion. In the case of unsatisfiability, it provides concrete counter-examples that are useful in analyzing why and how the conflict happened.

We model the IoT system as a Finite State Machine (FSM), such that detecting and resolving a conflict can be reduced to a property-checking problem on the FSM, which can be solved using any existing model checkers. The FSM can be seen as a directed graph since nodes represent any subsystem's internal states and the arrows represent state transitions. Thus, the model checking amounts to checking whether this graph fulfills (is a model of) the property to be checked.

D. Static vs. Dynamic Verification

It would be ideal for checking for and removing all conflicts across IoT subsystem interactions statically at system design time. However, this is impossible in reality for three reasons: (a) most subsystems are deployed incrementally rather than all together, (b) the operation of the subsystems continues to evolve as a result in device addition/removal/upgrade and software changes and (c) a static checking must consider all possible interactions. The last part is crucial since the only way to rule out conflicts is to consider the most extreme scenarios where most of the possible conflicting operations happen concurrently – a situation that may have no relevance to reality.

The other extreme is checking dynamically when critical actions are requested or when conflicting operations are actually attempted. Much of the past work considers this situation and must resolve the detected conflict by taking immediate action, usually blocking or delaying the less important activities. Thus, proactive conflict avoidance is no longer possible.

In this paper, we explore intermediate situations, where we look ahead by some time duration τ and examine the likely events based on the current conditions. We could then resolve them proactively by changing the thresholds and durations, if possible. These changes could be temporary by default but may also be made permanent based on human input. If an unanticipated event does occur, it can be handled as in normal dynamic resolution, but the look ahead should handle most events much more flexibly. The events that frequently occur under certain conditions (e.g., energy management related events under low or high outside temperature) can be learned either by the system itself over time or by a separate ontological reasoning model. For example, the energy management subsystem can quickly know that during winter, heating is on but AC is not, but it is harder to learn that the load shedding signal is unlikely to be generated during winters.

Note that "looking ahead" does require the theories to include the physics of the processes involved (e.g., heat and mass transfer, illumination, etc.). Still, in many cases, simple approximations should be adequate. The duration τ above can be chosen by considering the tradeoff between the reliability of prediction and the flexibility in resolving the conflicts.

IV. Automated Conflict Resolution

Given the rules and safety properties (SPs) in LTL notation, we can use a model checking tool to find conflicts and generate a minimum UNSAT core that contains all the conflicts. For each conflict, we need to first trace down and find all the rules where a change could flip the truth value. This provides us with the rules that can potentially be weakened to resolve the conflict. The key challenge is to determine which underlying semantic variables should be changed in the rule conditions and by how much so as to resolve the conflict. There are two methods to accomplish this, which we discuss next.

A. Weighted Partial Maxsat Based Approach

A special case of constrained optimization problem called the *Weighted Partial Maxsat* (WPM) has attracted a lot of attention in the literature; for example, the recent book [12] contains a survey of several methods. The solution approaches include both "complete" and "incomplete" methods. The former refers to linear search or branch and bound methods that always produce the definitive answer, whereas the latter refers to combinatorial optimization approach that may not always produce an answer in the limited iterations allowed.

Unfortunately, we cannot use these algorithms directly since the problem is stated with Boolean variables only. For example, consider the following rules where "temp and "Lum" refers to the temperature and luminance in the room. These two rules can conflict if both preconditions are true.

$$R_1 : \mathrm{Lum} < 200 \wedge \mathrm{open\_shade} \Longrightarrow \mathrm{shade\_open} \qquad (4)$$

$$R_2 : \mathrm{temp} > 85F) \wedge \mathrm{shut\_shade} \Longrightarrow \mathrm{shade\_shut} \qquad (5)$$

These rules can be translated into the following Boolean expressions

$$R_1 : \neg(x_1 \wedge x_3) \vee x_4, \quad R_2 : \neg(x_2 \wedge \neg x_3) \vee \neg x_4 \qquad (6)$$

$$\text{where } x_1 = \mathrm{Lum} < 200, \quad x_2 = \mathrm{temp} > 85F, \qquad (7)$$

$$x_3 = \mathrm{open\_shade}, \quad x_4 = \mathrm{shade\_open} \qquad (8)$$

We could now speak of the satisfiability of $R_1 \wedge R_2$ in (6) purely in terms of assignment of true/false to x_i's. In the conflict mitigation context, the problem is to flip the fewest x_i's to make $R_1 \wedge R_2$ satisfiable. We can also associate costs with flipping x_i's and thus consider cost minimization as the

objective. The problem could then be considered as an instance of the WPM problem. However, this approach's problem is that we lose the underlying semantics of x_i's and may come up with a practically meaningless solution. Normally, the WPM literature assigns fixed costs to individual clauses in the formula's CNF representation. In contrast, here, we need costs associated with how much we perturb the underlying state variables. Therefore, we do not use WPM approaches in this paper.

B. Combinatorial optimization Based Apporach

We adopt a combinatorial optimization approach to bring minimum changes to the rules to make them conflict-free. We first pass the set of rules **R** through a MAXSAT solver, which returns an UNSAT core **C** in case of conflicts. The UNSAT core consists of conflicting rules and safety properties. The next step is to find ways of "weakening" the rules.

These methods move from one solution to the next while keeping track of the best solution found so far. The next solution is found by perturbing some of the variables in the existing solution so that the solution does not get stuck around a local optimum. This is generally accomplished by making both large random jumps in the state space and local searches for better solutions.

Combinatorial optimization is generally described as unconstrained (except for simple constraints like bounds on values of variables) for which numerous algorithms are available [13], [14]. Of these, while Genetic Algorithm (GA) is best known, it is often the worst performing whereas variants of simulated annealing (SA), called very fast simulated annealing [15] have been quite successful on many problems [16]. One algorithm that works extremely well for high-dimensioned problems is called DDS (Dynamically Dimensioned Search) [17]. It initially perturbs most of the variables but perturbs fewer of them as the iterations proceed, akin to temperature decrease in SA. We make use of DDS in this work.

Our problem is a constrained optimization problem where the constraint is the satisfiability of the altered formula. Several methods have been explored for this as well as detailed in the survey paper [18]. It is shown that merely moving from one feasible solution to the next (i.e., entirely ignoring intermediate infeasible solutions) is a poor strategy; instead, it is best to use a method that attempts to balance between feasibility and optimality when moving around in the state-space. In this paper, we use the epsilon-constraint satisfaction method, which appears to work quite well in practice. This method essentially quantifies the notion of "degree of feasibility" based on how close the proposed solution is to the constraint boundaries. This is then used along with the current cost to decide how to perturb the parameters.

C. Intelligent operational rule perturbation

The key to finding a near-optimal solution quickly in combinatorial optimization algorithms is the use of *domain knowledge*. The perturbations to the solution made at each stage represent various correlations and dependencies. To this end, we classify the safety properties into many categories and the kind of perturbation of operational rules that can resolve the conflict with them.

In the following, we state these categories in negative form, i.e., describe them as a pair of temporally proximate events that *should not* happen. We use the notation (C,A,X,E,t_e) to denote controller C executing action A on actuator X, which results in an effect (or change in some relevant state variables) denoted E, and this effect has persisted for the last t_e time units. The effect E may concern a Boolean variable (that becomes flipped from false to true or vice versa) or an arithmetic variable whose value is driven above or below some limit. In the examples below, "EM" means energy management, "LM" means lighting management, "FM" means fire-management, and "WM" means water-management. We assume that the first of the two mentioned events happens first for convenience in describing the conflict and its mitigation.

Type 1a: An actuator that has two logically opposite states, receives conflicting commands from two different controllers within a short time-period, e.g., (EM, Close, door, $-$, δ) and (FM, Open, door, $-$, 0) where δ is below some threshold $\delta_{\min}$. Here the effect part could be different for the two systems but does not matter (and hence not mentioned).

Type 1b: This is a variant of 1a where the control action has multiple states as described by the effect part. The conflict could now arise because two controllers have opposing effects within a short time-period, e.g., (EM, Rotate, shades, angle=60, δ) and (LM, Rotate, shades, angle=0, 0) where δ is below some threshold $\delta_{\min}$.

Type 2: An actuator when actuated in a specific way has opposite impact on two subsystems the e.g., (WM, turn_on, Sprinkler, floor_wetness>1cm, δ) and (FM, turn_on, Sprinkler, extinguish_fire, 0) where δ is above some threshold $\delta_{\max}$.

Type 3: Two different actuators receive commands (from their controllers) that have an opposite effect on some state variable, e.g., (EM, turn_on, Heater, temp>75F, δ_1) and (FM, Open, Window, temp<65F, δ_2) where $\delta_2 < \delta_1$. The example corresponds to the scenario where both the "Heater-on" and "window-open" conditions overlap.

We now discuss how the above 3 types of conflicts can be mitigated by changing the thresholds.

Type 1: This conflict can be resolved by making $\delta > \delta_{\min}$. If this is done by perturbing the timing of the two conflicting events, the perturbations must be properly correlated. In particular, if the first event is preponed, the second event should not be preponed. Similarly, if the first event is postponed, the second event should be postponed by larger than this amount.

Type 2: This conflict can be resolved by making $\delta < \delta_{\max}$. The perturbation restrictions are complementary that for Type 1 conflicts. If the first event is postponed, the second event should not be postponed. Similarly, if the first event is preponed, the second event should be preponed by larger than this amount.

Type 3: This conflict is more difficult to handle since the overlap can be avoided in only two ways: (a) blocking the

second event outright, or (b) postponing the second event until the opposite action for the first event has taken place.

D. Domain knowledge based DDS

From the UNSAT core **C** we identify the rules **R'**$\subseteq$**R** that are directly or indirectly dependent on **C**. The dependency is expressed via a dependency graph G, where the vertices denote the rules/safety-properties, and the (directed) edges denote the dependency between them. We then take the transitive closure of G (say G') using the Floyd–Warshall algorithm. Thus, in G' an edge $i \rightarrow j$ denotes that j is directly or indirectly dependent on i.

From G' we mark the vertices that are in **C** or their neighbors, to find **R'** wherein we tune the tunable parameters, such as different thresholds and time variables to eliminate the conflicts (if possible). In case of conflicts that do not involve threshold or time variables, we resolve the conflicts by allowing the higher priority event to execute the action at run time. The term "priority" refers to the cost of not taking action prescribed by the operational policy. For example, in a smart home environment, the fire and safety controller issues a command to open all doors and windows in case of fire. Simultaneously, due to a drop in outside temperature, the climate controller gives a command to close all doors and windows. Such a conflicting scenario does not involve any time or threshold variable, and hence the priority takes precedence. In this case, since the fire event takes higher priority, the fire and safety controller's command gets executed at the run time. The overall optimization problem can be described as follows.

$$\begin{aligned} &\text{Min}\,\mathbb{M}\left(\sum_i (\tau_i - \hat{\tau}_i)^2 + \sum_i \left(t_i - \hat{t}_i\right)^2\right) + \mathbb{N} \\ &\text{s.t.}\,\mathbb{R}'\left(\hat{\tau}_i, \hat{t}_i\right) \end{aligned} \tag{9}$$

where τ_i, $\hat{\tau}_i$ are the thresholds and modified thresholds respectively, whereas t_i and $\hat{t}_i$ are the timings and modified timings, respectively. $\mathbb{N}$ is the number of clauses where τ_i or t_i has changed to $\hat{\tau}_i$ or $\hat{t}_i$ respectively. $\mathbb{M}$ is a large number that is at least as large as the number of clauses in UNSAT. Thus, when the tuning cost is identical, then the objective function chooses the solution with the minimum number of changed clauses.

Consider an objective function $f(x)$ of input vector x, along with a set of constraints $\phi_i(x), i = 1,2,...,K$. Let $\sigma_i(x) \in [0..C_i]$ denote a cost measure for constraint $\phi_i(x)$ that indicates to what extent the constraint is violated for input vector x. By definition, if the constraint is satisfied, then $\sigma_i(x) = 0$. Let $\sigma(x) = \sum_{i=1}^{K} \sigma_i(x) / \sum_{i=1}^{K} C_i$ denote the overall normalized cost of violating the constraints, which has the range $[0..1]$. Now consider an existing solution x_1, and new proposed solution x_2. The ε comparison between them defines a specific way of determining if x_2 is better than x_1 by considering both the objective function and the constraints. In particular, suppose that the objective is to minimize the objective function. Then the "epsilon less than" relationship between x_2 and x_1, denoted $x_2 <_\varepsilon x_1$ is defined as follows: [18], [19]:

$$x_2 <_\varepsilon x_1 = \Leftrightarrow \begin{cases} f(x_2) < f(x_1), & \text{if } \phi(x_2),\ \phi(x_1) < \varepsilon \\ f(x_2) < f(x_1), & \text{if } \phi(x_2) = \phi(x_1) \\ \phi(x_2) < \phi(x_1), & \text{otherwise.} \end{cases} \tag{10}$$

The intuition behind this comparison is that, if the solutions x_1 and x_2 are feasible, slightly feasible (as determined by ε), or having the same sum of constraint violations (the number of unsatisfied constraints in our case), then they are compared using their objective values $f(x_1)$ and $f(x_2)$. Otherwise, if both x_1 and x_2 are infeasible, they are compared based on their sum of constraint violations. The overall scheme is discussed in Algorithm 1.

Algorithm 1 Proposed solution of ε-DDS

1: r: Neighborhood perturbation size {INPUT}
2: M: Maximum number of evaluation {INPUT}
3: $\mathbf{x_0} = \{x_1, x_2, \ldots, x_d\}$: Initial solution {INPUT}
4: $\mathbf{x^{min}}$: Lower bound of decision variables {INPUT}
5: $\mathbf{x^{max}}$: Upper bound of decision variables {INPUT}
6: $\mathbf{x^{best}}$: Best solution found {OUTPUT}
7: **procedure** ε-DDS:
8: $\mathbf{x^{best}} = x^0$, $F^{\text{best}} = F(x^0)$;
9: **for** i = 1 to M **do**
10: Randomly select J of the D decision variables with a probability $P(i) = 1 - \ln(i)/\ln(M)$, and include it in $\{N\}$;
11: For $j = \{1,2,\ldots,J\}$ variables in $\{N\}$, use the domain knowledge to perturb x_j^{best}, i.e. $x_j^{\text{new}} = x_j^{\text{best}} + \sigma_j \mathbb{N}(0,1)$, where $\sigma_j = r\left(x_j^{\max} - x_j^{\min}\right)$;
12: **if** $F(\mathbf{x^{new}}) <_\varepsilon F^{\text{best}}$ **then**
13: $F^{\text{best}} = F(\mathbf{x^{new}})$, $\mathbf{x^{best}} = \mathbf{x^{new}}$;
14: **end if**
15: **end for**
16: Return F^{best}, $\mathbf{x^{best}}$;

Thus in every iteration in the ε-DDS, the scheme identifies a set of J decision variables and includes them to $\{N\}$. Next, we perturb $\mathbf{x^{best}}$ using the following expressions

$$x_j^{\text{new}} = x_j^{\text{best}} + \sigma_j \mathbb{N}(0,1), \text{ where } \sigma_j = r\left(x_j^{\max} - x_j^{\min}\right) \tag{11}$$

while respecting the domain knowledge as discussed in section IV-C. If the new solution is $F(\mathbf{x^{new}}) <_\varepsilon F^{\text{best}}$, then we take $\mathbf{x^{new}}$ as the best solution, continue the same process until the stopping criteria is reached.

V. Experimental Evaluation

A. Smart-home emulation

In order to experiment with a wide variety of processes and events in a smart home, we use a comprehensive smart home emulator called Home-IO, built by CReSTIC laboratory [20]. The simulator is a virtual house's real-time simulation software that can modify the environment and automation level. It supports third-party devices, software (Python, Matlab, etc.), and hardware (microcontrollers, programmable logic controllers, Raspberry Pi, etc.). Home-IO also has several advanced physics modeling capabilities; for example, it can realistically model changes in environmental parameters such as temperature, luminance, etc., based on the house's geographical coordinates.

TABLE I: Operational rules for different controllers

ID	Rule Description
	Operational rules for Light controller
R1	For any area A, every ten minutes the luminance value is calculated using daylight factor function
R2	If the luminance for any area A is less than a threshold (B_0), then it is night
R3	If the luminance for any area A is greater than the threshold (B_0), then it is day
R4	The lights for any area A must be turned on when a motion is detected and luminance value is less than a threshold (B_1)
R5	The lights for any area A must be turned off after 2 minutes, when there is no motion is detected, and luminance value is greater than or equal to the threshold(B_1)
R6	If the luminance for any area A is less than a threshold (B_2) and if it is day then open the roller shades
R7	If the luminance for any area A is greater than a threshold(B_2) and if it is day then close the roller shades
R8	The value of threshold B_2 is greater than B_0
R9	If the motion_detected for any area A is more than 3 times then, user is present in the area
R10	If no motion_detected for any area A for more than 5 minutes, user is not present in the area
R11	When the user is present in area A for more than 5 minutes then user_present mode is activated
R12	When the user is not present in area A for more than 10 minutes then user_away mode is activated
	Operational rules of Fire controller
R13	After two minutes the fire is detected, fire-alarm is turned on
R14	Once the fire is detected, the smoke level rises and reaches a maximum threshold *'t'* within 5 minutes
R15	Once the fire is detected, The room_temperature rises steadily by 35F every minute
R16	The doors(main door, patio) and windows must be open and unlocked until the smoke_level is less than threshold *t*
R17	Turn off the fire_alarm three minutes after it is turned on
R18	If the room temperature is greater than or equal to 155F then sprinkler_head is activated and water_pump is open
R19	Once the sprinkler_head is activated, The room_temperature drops by 35F every minute
R20	If the room temperature is drops below 100F then sprinkler_head is deactivated and water_pump is closed
R21	When the water_pump is open, the water_flow is at a constant rate of 225litre per minute(a constant *C*)
R22	When the water_pump is closed, the water_flow is zero
R23	Once the sprinkler is on, turn off the sprinkler after 4 to 6 minutes
	Operational rules for Climate controller
R24	The room_temperature is calculated every ten minutes using heat_transfer function
R25	When the room_temperature $\geq$ 75F then turn on cooler and close all doors and windows
R26	When the room_temperature $\leq$ 68F then turn on heater and close all doors and windows
R27	The outside_temperature is calculated every ten minutes using outside_air_temperature function
R28	If the outside_temperature drops below 0F, then load_shedding signal is turned on
R29	When the load_shedding signal is turned on then turn off the hvac or turn off the water_pump
R30	When the heater is on, the room temperature rises by 35F every 30 minutes
R31	When the cooler is on, the room temperature drops by 35F every 30 minutes
R32	If the room_temperature is in between 66F and 71F then turn off the cooler after 3 to 5 minutes and open windows and patio door
R33	If the water_flow is 0, then the water_level is 0
R34	If the water_flow is at constant rate *C* for four minutes then the water_level reaches a threshold *l*
R35	If the water_level is above threshold *l* then, the flood_sensor is activated and water_pump is closed and open all doors and windows
R35	The flood_sensor is deactivated after two minutes and evaporation starts with a constant rate of *V* until the water_level reached zero
	Operational rules for Security controller
R37	The doors(main door, patio) must be closed and locked when the door sensor is inactive
R38	The doors (main door, patio) and windows must be closed and locked until the user is not present
R39	The garage door must open when motion is detected and when valid_keypad_code is entered
R40	The valid_keypad_code is a predefined four digit numerical code (a constant *K*)
R41	The garage door should remain closed when wrong keypad_code is entered by user and when there is no motion detected
R42	When garage door is opened then unlock the main door
R43	The intruder alarm should turn on if wrong keypad_code is entered more than three times for accessing main door or garage door
R44	The main door should open when a valid_keypad_code is entered
R45	The intruder alarm should go off after 3 minutes
R46	When intruder alarm is on, then lock main door, patio doors and windows
	Operational rules for Surveillance controller
R47	For any area A, Until the user_present mode is activated the security_cameras (indoor_camera,outdoor_camera) should record video for 't' time slots
R48	The video recorded is analyzed for any unlawful activity
R49	In case of unlawful-activity exhibited by user, the security alarm turns on
R50	The security_alarm is turned off after two minutes

Home-IO provides a simplified model to simulate real-time heat transfer, including radiation, convection, and conduction

TABLE II: Safety Properties

ID	Rule Description
S1	When the sprinkler is on for three minutes, then the water valve is off
S2	When the fire alarm is on for more than 1 minute, main door, patio doors and window are opened for 8 to 10 minutes
S3	The windows must not be unlocked and open when the heater or cooler is on
S4	When the heater or cooler is on, the main door and patio door should not be open for more than 5 minutes
S5	If the garage_door is opened for more than 2 hours then close the garage_door
S6	During night if the fire alarm is on then turn on the lights for 3 to 4 minutes
S7	The AC and the heater must not be on at the same time
S8	All the doors and windows must be closed within 2 minutes when user_present mode must is activated

phenomena. Exchanges between different air masses are simulated. Door and window opening/closing affect the temperature as in a real house due to temperature differential between outdoors and indoors and the airflow. The wind facilitates the transmission of heat between the house and outside air. The upwind house walls are more reactive to outdoor heat transfer. Cloud cover decreases sun and sky radiation effects. Eventually, humidity changes the dew point, which affects how the outside air affects the house temperature. The home has multiple rooms, each of which has facilities for smoke detection, carbon monoxide detection, smart lights, smart window blinds, smart doors, etc. Therefore, the house is a "black box," a data source, and a place for experiments.

Because of all these features, we believe that the use of Home-IO is far more powerful than a real smart home controller (e.g., Samsung SmartThings [21]), where the readings of various sensors or actuators would be either extremely limited or artificial (e.g., pretending that there is a fire). Our control and analysis logic was implemented in Python using visual studio 2020 (version 1.46) installed on an Intel(R) Core(TM) i7-7700 CPU @ 3.60 GHz, 32 GB RAM, and 1 TB SSD.

Fig. 1(a) shows our simulated scenario using Home-IO. For simplicity, we consider only two rooms in the home, namely the kitchen and living room (LR), which have only 20 out of a total of 112 various IoT devices. Operational rules for automation of these devices and the conflict detection, resolution algorithm are implemented as appropriate Python routines.

B. Emulated IoT system in Home-IO

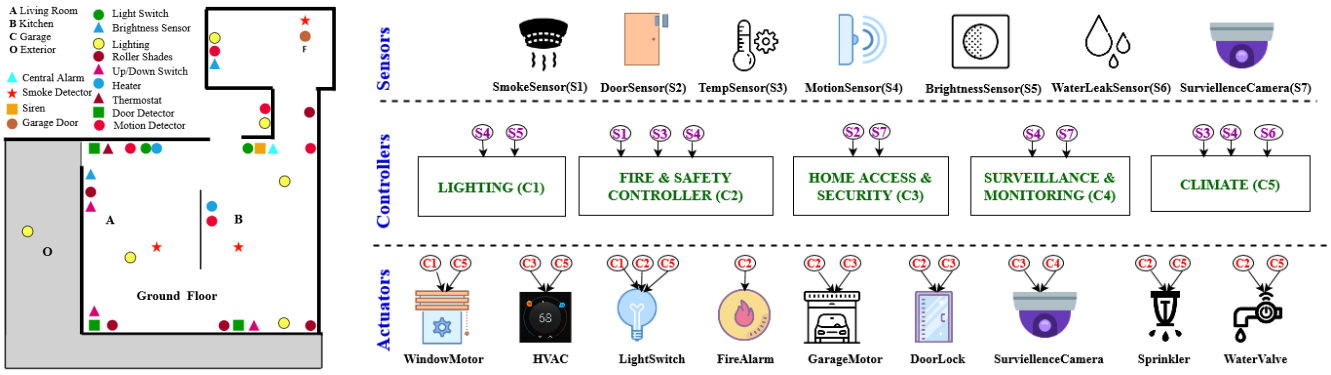
In this section, we introduce the motivation of our work by presenting a moderate scale IoT scenario deployed in a typical IBMS, as shown in Fig. 1(b). The system comprises five controllers running the following dedicated services:

(a) *Lighting Management*, on controller C1, that manages the lighting of the connected devices providing perfect luminance settings inside an IBMS.

(b) *Fire and Safety Management*, on C2, provide safety for residents in case of fire.

(c) *Security Management*, associated with the controller, C3 that controls the entry or exit of users, further preventing unauthorized access.

(d) *Surveillance and Monitoring*, on controller C4, monitors the unlawful activities of people inside and outside the IBMS.



(a) (b)

Fig. 1: (a) Virtual floor plan created in Home-IO. (b) Different components used for the simulations and their interactions.

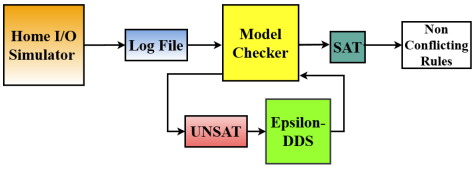


Fig. 2: System overview implemented in Home-IO

(e) *Climate Control*, is a subsystem on C5 that sets out energy-saving features and user convenience along with controlling and maintaining ambient climate service.

The subsystems interact with one another when operating in a shared space. For convenience, we designed the interaction between the smart devices within and across subsystems as routines. A routine may be either user-initiated or event/time-triggered. Our simulated model allows full flexibility to set complex routines, including components for catching and responding to concurrency conflicts and safety property violations.

Fig. 1(b) shows the interactions between the controllers, sensors, and actuators. Note that all actuators other than the fire-alarm can be operated by more than one controller. However, the preconditions and the purpose differ widely from one service to another. For example, the energy management controller uses the light switch for saving power, whereas the safety management controller takes over control during an emergency. For simplicity, we assume that the sensor readings themselves are universally visible to all controllers.

Based on the scenario presented in Fig. 1(b), we develop 50 operational rules that are specific to each controller and verify them against five types of safety properties defined in section IV. Various rules and safety properties are listed in the Tables I-II. We only list the textual version of the rules for readability and space reasons, rather than the actual LTL version used for the implementation. As an illustration, rule R16 and R35 can be translated to LTL using the until (U) operator as follows:

$G[X^1(\forall_i \text{ doors}_i\text{=open} \wedge \forall_i \text{ windows}_i\text{=open U (smoke\_level} < t')]$

$G[X^2(\text{flood\_sensor = activated} \wedge \text{deactivate\_flood\_sensor} \implies \text{flood\_sensor = deactivated} \wedge \text{evaporation}(v) \text{ U (water\_level = 0)}]$

C. Implementation of ε-DDS in Home-IO

Fig. 2 shows our overall approach of how the existing Home-IO emulator is modified to implement ε-DDS. Operational rules, safety properties, and device descriptions are fed into the Home-IO simulator. The log file provides details of each event, the event timestamp, the event's duration, and the actions taken. It is then used for modeling interactions between devices and rules to build a Finite State Machine (FSM). In an FSM, the device states and operational rules correspond to the state and transactions, respectively. The *state* can be represented using the values of features/attributes of each device. For example, a thermometer may have a pair of attributes, namely time and temperature, where the temperature reflects the value measured at the specified time. A device's state may be changed via measurement action (in the case of a sensor device) or by external input (in the case of an actuator).

The inputs to the model checker are (i) Operational rules and safety properties in LTL, and (ii) Specification of every IoT component (e.g., sensors, actuators) in the form of a transition system built from the log file. The FSM is verified against a model checker to identify the conflicts. In case the model-checker finds some violation of safety properties, it provides a counter-example that denotes the unsatisfiable (UNSAT) clauses that lead to the property's violation. Our proposed algorithm ε-DDS then takes the UNSAT clauses and perturbs the time or threshold variables intelligently to resolve conflicts. The output of ε-DDS finally returns a set of satisfiable clauses that indicate the conflicts are resolved.

D. A case study

For the case study, we pick up a case that starts with the rules <R1, R2, R4, R5, R9, R11, R24 to R31, R44> that are non-conflicting, and then add the conflicting rules <R13, R16, R18, R21, R23, R35, R43, R47>. Beginning with the event, "fire detected in the kitchen", multiple things will happen as shown in Table III.

Based on all these violations of safety properties, the model checker returns a UNSAT. After that, our conflict resolution scheme perturbs the following thresholds: (a) kitchen temperature threshold, (b) LR temperature threshold, (c) smoke level threshold, (d) water-level threshold, (e) time duration of

TABLE III: Sequence of Events Due to Kitchen Fire

1	Smoke level rises ⇒ Smoke detector turns on after 2 minutes ⇒ Smoke detected.
2	Kitchen temperature rises up to 100F.
3	Sprinkler head is activated, and sprinkler is turned on ⇒ Water pump is turned on.
4	Water flows at a constant rate (rule R21) but due to the low evaporation rate, water level threshold is reached before the fire is extinguished.
5	Consequently, floor flooding sensor is triggered, and the water pump is off. This creates a "conflict" which can be resolved by changing the water-level threshold.
6	Because rules R15 and R25 are triggered, the kitchen AC is switched on as the temperature rises above 90F. The AC will remain on until the temperature goes down to 70F.
7	The high smoke level causes the LR and kitchen windows to be opened, which will stay open until the smoke level drops below a threshold as defined by rule R16.
8	LR temperature drops to 67F ⇒ Rule R26 turns on the heater ⇒ safety properties S4 and S7 are violated.
9	Due to nightfall, rule R4 turns off the lights, but safety property S6 states that the lights must be on for 3-4 minutes in case of fire at the night, creating a conflict.
10	Rule R11 implies that the user is home ⇒ safety property S8 requires that the doors/windows be closed within 2 minutes. But this is not possible due to R16 ⇒ S8 is violated.
11	Meanwhile, an intruder enters successfully (using guessed main door key-code) ⇒ R43 triggers intruder alarm.
12	User present ⇒ Surveillance cameras disabled, but intruder alarm on ⇒ Record video. Here the conflict is resolved by priority-based resolution.

Solution 1: 4.10, 69.4, 70.0, 5.1, 3.9, 4.5
Best objective function value of 94.617664 found at Iteration 180
Time of execution for Trial 1 was 0.0172460.018429 seconds or 0.000005 hours.

Solution 2: 3.5, 73.4, 74.0, 5.4, 4.3, 4.05
Best objective function value of 101.675437 found at Iteration 195
Time of execution for Trial 1 was 0.018429 seconds or 0.000005 hours.

After running Model Checking: No UNSAT clauses in Solution 1 and Solution 2
Best Solution is: Solution 1

Fig. 3: Results from $\varepsilon-$DDS Algorithm

window and door opening, and (f) motion-detection threshold. These decision variables are passed through the ε-DDS algorithm. The results are shown in Fig. 3. The results show that we have two solutions along with the new values of decision variables, and based on equation 10, the best solution is selected. With updated rules passed through the model checker, satisfiability is achieved.

The case study shows multiple asynchronous events, all occurring concurrently to create numerous conflicts for illustration purposes. Suppose in our lookahead mechanism, we can establish, based on the prevailing conditions, that certain events cannot co-occur or are not very likely. In that case, we can limit the conflicts and hence the need for perturbations. As mentioned earlier, this also depends on the time-horizon chosen. If we look ahead by only 30 minutes, several of the co-occurring events can be considered very unlikely, but a 4-hour window will make several of them worth considering. If we consider a 4-hour lookahead window without considering the moderately likely events, we will end up with more priority based resolutions.

E. Evaluation results

To evaluate the performance of ε-DDS, we implemented our algorithm along with Home-IO and satisfiability solver NuXMV on a desktop with Intel(R) Core(TM) i7-7700 CPU @ 3.60 GHz, 32 GB RAM, and 1 TB SSD. We start with a baseline system of 50 operational rules and safety properties (hereafter called "rules"). We generate many conflicting scenarios for this, as described in the case study. The system includes 19 rules with perturbable thresholds. We consider three possible perturbations of each of these rules, thereby yielding a total of 57 different possible perturbations. For these experiments, we use simplified physics as mentioned in section V-A so that the processing time of a case is mostly dominated by satisfiability checking rather than by physics.

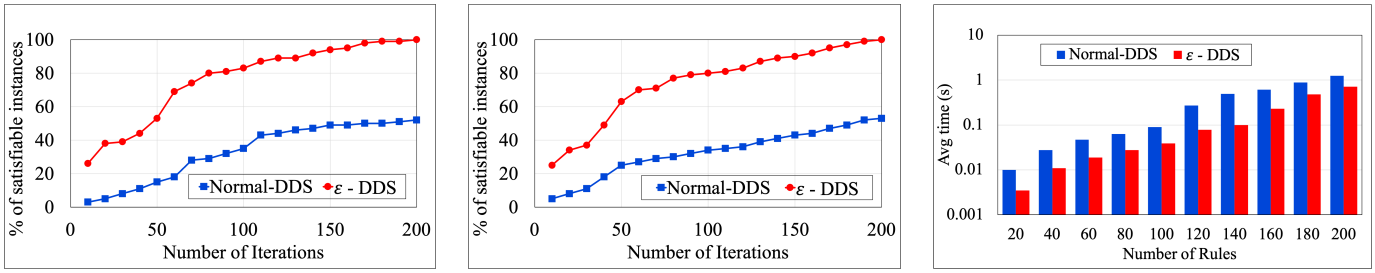
Fig. 4(a) shows the results that are averaged over 57 such cases. The x-axis indicates the number of iterations needed by combinatorial optimization. It is limited to a maximum of 200, whereas the y-axis shows the percentage of cases where the algorithm could resolve the conflicts. It can be seen that our algorithm, powered by domain knowledge, can resolve the conflicts in 100% of the cases in about 175 iterations. In contrast, the normal-DDS, without any domain knowledge, can fix only 53% out of the test cases at 200 iterations.

We further evaluate our algorithm's performance where we create somewhat different conflict-causing scenarios by adding more operational rules. We first begin with a set of non-conflicting rules and then add more operational rules to create 29 different conflicting scenarios. Fig. 4(b) shows the comparison of normal-DDS and ε-DDS for these 29 scenarios. It can be seen that the Normal-DDS algorithm, due to lack of domain knowledge, manages to resolve the conflicts for only 54% of the total generated conflicting scenarios after 200 iterations. In contrast, our algorithm can fix all of the conflicting scenarios.

It is essential to note that the change in thresholds may not resolve the conflict permanently and instead only postpone its occurrence. For example, if a rule says that the window must be shut because the person is at home, and the temperature continues to rise, it may cross the increased threshold perturbed to resolve the conflict. From a practical perspective, such situations become increasingly unlikely as thresholds are altered, but the potential conflict cannot be entirely ruled out. The ultimate resolution may then occur either by traditional means (e.g., priorities among actions) or by some action that is entirely outside the behavior captured by the formal model.

Fig. 4(c) shows the comparison of normal-DDS and ε-DDS in terms of processing time as the number of rules increases. Notice that the y-axis is in log-scale; therefore, the advantage of ε-DDS in terms of speed is quite substantial. This figure shows that our approach to dynamic conflict detection and resolution can comfortably handle rather large systems regardless of whether the resolution is done proactively (by changing thresholds) or reactively (by using relative priorities or cost).

Fig 5 shows how the running time of our algorithm varies



(a) (b) (c)

Fig. 4: Comparison of ε-DDS & normal-DDS: (a) and (b): % of satisfiable instances by random perturbation and by adding rules, (c) average running time

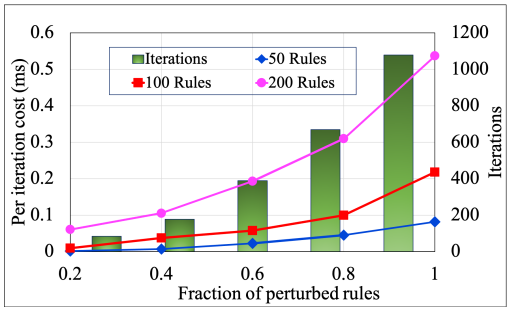


Fig. 5: Num of iterations and per iteration cost vs. fraction of perturbed rules

with the number of rules and the fraction of those rules that are eligible for perturbation. For this, we consider two additional cases with 100 and 200 rules (in addition to the original 50). Using the same scheme for perturbation as described above, we have a total of 57, 114, and 228 possible perturbations in the 50, 100, and 200 rule cases, respectively. We consider varying fractions of these possible perturbations in Fig 5, i.e., for the case of 100 rules, the fraction 0.4 means that (0.4×114) perturbations are considered.

The left Y-axis corresponds to the per-iteration cost, which is seen to grow both with the fraction of perturbable rules and the total number of rules. The growth is nonlinear but relatively slow. The number of iterations, shown via bars with the scale on the right Y-axis, also grows non-linearly but at a somewhat quicker pace. Since our tests were carried out on a relatively modest consumer desktop, we believe that the mechanism can quickly scale to thousands of rules with a well-equipped server. This, of course, assumes a simplified physics, which should be adequate in most cases. More precise physics based emulation would require more computing power in extensive systems.

VI. Conclusions

In this paper, we have examined the problem of conflict detection and mitigation in large IoT systems by introducing the notion of "safety properties" (SPs) that could potentially conflict with operational rules (ORs) of the system. We allow both the ORs and SPs to include time and temporal logic operations as an integral part. We allow the model to "look ahead" to proactively catch most (though not necessarily all) potential conflicts and mitigate them by temporarily perturbing the various thresholds or durations in the rules, if possible. Such an approach offers much more flexibility in dealing with conflicts than resolving the conflicts reactively by blocking or delaying one of the conflicting actions. We show that our mitigation approach, based on intelligent combinatorial optimization, can resolve the conflicts in 100% of the cases, as long as such a resolution is feasible and easily scales to hundreds of rules or more.

References

[1] C. Vannucchi *et al.*, "Symbolic verification of event–condition–action rules in intelligent environments," *Journal of Reliable Intelligent Environments*, vol. 3, pp. 1–14, 02 2017.

[2] Y. Sun *et al.*, "Conflict detection scheme based on formal rule model for smart building systems," *IEEE Transactions on Human-Machine Systems*, vol. 45, no. 2, pp. 215–227, 2015.

[3] M. Ma *et al.*, "Cityguard: A watchdog for safety-aware conflict detection in smart cities," in *IoTDI*, 2017, pp. 259—-270.

[4] M. Ma *et al.*, "Detection of runtime conflicts among services in smart cities," in *IEEE SMARTCOMP*, 2016, pp. 1–10.

[5] S. Munir *et al.*, "Depsys: Dependency aware integration of cyber-physical systems for smart homes," in *ACM/IEEE ICCPS*, 2014, pp. 127–138.

[6] F. Corno *et al.*, "Design-time formal verification for smart environments: an exploratory perspective," *Journal of Ambient Intelligence and Humanized Computing*, vol. 5, pp. 581–599, 2014.

[7] A. Al-Farooq *et al.*, "A formal method for detecting rule conflicts in large scale iot systems," *IFIP/IEEE IM*, 2019.

[8] A. K. Sikder *et al.*, "Aegis: A context-aware security framework for smart home systems," in *ACSAC*, 2019, pp. 28—-41.

[9] L. De Moura *et al.*, "Satisfiability modulo theories: Introduction and applications," *Commun. ACM*, vol. 54, no. 9, pp. 69–77, 2011.

[10] L. de Moura *et al.*, "Z3: An efficient smt solver," in *TACAS*, C. R. Ramakrishnan *et al.*, Eds., 2008, pp. 337–340.

[11] R. Cavada *et al.*, "The nuxmv symbolic model checker," in *CAV*, 2014, pp. 334–342.

[12] H. Hoos *et al.*, *Stochastic Local Search: Foundations and Applications*. San Francisco, CA, USA: Morgan Kaufmann Publishers Inc., 2004.

[13] M. Barrette *et al.*, "Statistical multi-comparison of evolutionary algorithms," *Bioinspired Optimizaiton Methods and their Applications*, 2008.

[14] M. Franchini *et al.*, "Global optimization techniques for the calibration of conceptual rainfall-runoff models," *Hydrological Sciences Journal*, vol. 43, no. 3, pp. 443–458, 1998.

[15] M. Vakil-Baghmisheh *et al.*, "A modified very fast simulated annealing algorithm," in *IST*, 2008, pp. 61–66.

[16] R. Arsenault *et al.*, "Comparison of stochastic optimization algorithms in hydrological model calibration," *Journal of Hydrologic Engineering*, vol. 19, no. 7, pp. 1374–1384, 2014.

[17] B. Tolson *et al.*, "Dynamically dimensioned search algorithm for computationally efficient watershed model calibration," *Water Resources Research*, vol. 43, no. 1, 2007.

[18] E. Mezura-Montes *et al.*, "Constraint-handling in nature-inspired numerical optimization: Past, present and future," *Swarm and Evolutionary Computation*, vol. 1, no. 4, pp. 173–194, 2011.

[19] T. Takahama *et al.*, "Constrained optimization by the epsilon constrained hybrid algorithm of particle swarm optimization and genetic algorithm," in *AI*, 2005, pp. 389–400.

[20] B. Riera *et al.*, "Home i/o: a virtual house for control and stem education from middle schools to universities," in *IFAC ACE*, 2016.

[21] "Samsung smartthings," https://www.samsung.com .