IoT

Review

Conflict Detection and Resolution in IoT Systems: A Survey

Pavana Pradeep and Krishna Kant *

CIS Department, Temple University, Philadelphia, PA 19122, USA; pavana.pradeep@temple.edu
* Correspondence: kkant@temple.edu

Abstract: Internet of Things (IoT) systems are becoming ubiquitous in various cyber–physical infrastructures, including buildings, vehicular traffic, goods transport and delivery, manufacturing, health care, urban farming, etc. Often multiple such IoT subsystems are deployed in the same physical area and designed, deployed, maintained, and perhaps even operated by different vendors or organizations (or "parties"). The collective operational behavior of multiple IoT subsystems can be characterized via (1) a set of operational rules and required safety properties and (2) a collection of IoT-based services or applications that interact with one another and share concurrent access to the devices. In both cases, this collective behavior often leads to situations where their operation may conflict, and the conflict resolution becomes complex due to lack of visibility into or understanding of the cross-subsystem interactions and inability to do cross-subsystem actuations. This article addresses the fundamental problem of detecting and resolving safety property violations. We detail the inherent complexities of the problem, survey the work already performed, and layout the future challenges. We also highlight the significance of detecting/resolving conflicts proactively, i.e., dynamically but with a look-ahead into the future based on the context.

Keywords: conflict detection; IoT system; safety properties; combinatorial optimization; linear temporal logic

Citation: Pradeep, P.; Kant, K. Conflict Detection and Resolution in IoT Systems: A Survey. *IoT* **2022**, *3*, 191–218. https://doi.org/10.3390/iot3010012

Academic Editor: Francesco Bergadano

Received: 18 January 2022
Accepted: 23 February 2022
Published: 28 February 2022

1. Introduction

1.1. Conflict Issues in IoT Systems

Large IoT systems are currently being deployed for smart management of many types of physical infrastructures, and many of these come together in a smart city (or, more generally, smart urban region) context. Recent surveys estimate that currently, the world will reach 43 billion WiFi-enabled gadgets by the year 2023 [1,2]. The vision of a smart city involves optimal management of every aspect of the city's needs, including electricity, water, sewer, trash, buildings, roads, subways, healthcare, goods delivery, etc. Designing, deploying, maintaining, and operating the IoT infrastructure needs for one such subsystem is a huge undertaking and will likely involve multiple organizations or "parties". It is even likely that different regions or parts of a city use various parties, resulting in quite a bit of heterogeneity even in a single subsystem. This issue becomes more complex when multiple IoT subsystems coexist in the same space, as they would in general. For example, various IoT subsystems may be deployed for lighting, Heating, Ventilation, and Air Conditioning (HVAC), fire safety, surveillance, etc. Such subsystems interact with one another by operating in the same shared environment, and their operation may conflict because each subsystem usually has little knowledge about the presence of others or how they behave.

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 special situations that may not or could not be anticipated or handled statically. This is particularly true when there are several independently designed/operated subsystems operating together. For example, suppose that we have Energy Management (EM), Lighting Management (LM), Fire Management (FM), and Water Management (WM) operating together. A conflict occurs when EM and LM attempt to close and open the shaded simultaneously (or within a short time period) to satisfy their objectives of saving energy and create adequate lighting.

1.2. Goals and Relationships to Other Surveys

The purpose of this paper is to provide a state-of-the-art survey of conflict detection and resolution in large IoT systems and point out the research challenges. We also consider the issue of cross-party access control, which is crucial in a multiparty environment but has not received much attention in the literature. The general area of applicability for this work is smart city cyber–physical systems, including management of buildings, people transport, transport of perishable (e.g., fresh food) and nonperishable goods, healthcare, etc., each of which may itself include multiple subsystems.

To the best of our knowledge, there are two survey papers on the topic. Ibrahim et al. [3] survey the related works in automation services authored by the occupants, particularly in building automation systems. Ibrahim et al. [4] conducted a comprehensive survey on conflict classification for IoT-based services/applications by analyzing the relationships using Satisfiability Modulo Theories (SMT). This article classifies interactions between IoT-based apps into two categories: (1) single-user IoT app conflicts detection and (2) multiple-user IoT app conflicts detection. Additionally, two different conflict categories are defined: (1) local conflicts, which occur between the rules and constraints of the same automation service or policy authored by the same user or admin, and (2) global conflicts, which occur during interactions between different automation services or policies. Based on local or global conflict types, the authors formalized the IoT automation services or policies and provided the definitions for these conflicts using a metric called Rule Satisfiability Measure (RSM), which indicates the state of the model checker representing SAT or UNSAT determined by the SMT solver. In contrast to this work, we take a broader view of the problem in terms of operating policies and safety properties that may be specified explicitly or implicitly. Additionally, we address all three types of conflicts: (a) static, which is detected via static analysis of the operational rules; (b) dynamic, which are detected at run-time when they actually occur; and (c) proactive/predictive, which are anticipated/predicted at run-time for a short period into the future based on the current context.

1.3. Paper Outline

The rest of the paper is organized as follows. Section 2 provides the background on the IoT system environment and discusses the issues of inter-dependencies and conflicts with an illustrative example. Section 3 provides a review of the literature on the various types of conflicts that can occur in an IoT system. Section 4 discusses the literature conflict identification or detection methods. Section 5 summarizes research on static vs. dynamic conflict detection and resolution strategies. Section 6 presents a proactive approach for detecting and resolving the conflicts via minimal cost alteration for the rules, and it also presents a comprehensive evaluation of the proposed algorithm. Section 7 discusses the future challenges in conflict detection and resolution. Finally, Section 8 concludes the discussion.

2. Dependencies, Conflicts, and Accessibility in IoT Systems

In this section, we discuss the key issues concerning conflicts and conflict resolution in large-scale IoT systems. Table 1 contains a list of abbreviations used throughout the paper.

Table 1. Table of Abbreviations.

Abbreviations	Meaning
ORs	Operational Rules
SPs	Safety Properties
ECA	Event Trigger Action
IFTTT	If This Then That
HNS	Home Network Systems
NLP	Natural Language Processing
FSM	Finite State Machine
SMT	Satisfiability Modulo Theories

2.1. IoT System Environment

IoT devices available on the market include most of the common household appliances, lighting, and healthcare products [1,5–10]. Even legacy or "dumb" devices can be controlled through products such as "smart things" [11] and "TWINE" [12]. Through remote sensing and control, IoT devices provide substantial improvements in operational efficiency, safety, and comfort, as in the case of the intelligent building management system (IBMS) [13–16].

An IoT subsystem includes a variety of sensors and actuator IoT devices managed by one or more controllers. It may also have some intermediate devices such as data aggregators (aggregating data from multiple sensors with the same functionality), traffic shapers (for threshold or intensity-based transmission), gateways, and protocol converters (converting between generic and device-specific commands), etc.

The automation capabilities provided by a controller can be classified into two categories: (1) IoT Application Services and (2) Set of Operation Rules (ORs). An IoT application service is a collection of automation rules that an end-user creates to customize the behavior of specific devices within a spatio-temporal domain of interest. Each service rule predominantly follows a trigger-action paradigm, where the IoT application responds to a trigger (for example, when the user enters the house) by commanding an action (e.g., turn on the coffee maker). Additionally, an app may send multiple actions to the same device or multiple devices. While a trigger can activate multiple apps concurrently, an action can also directly or indirectly activate other apps. Given that any typical IoT system is equipped with numerous such apps, the design principle underlying this paradigm can result in conflicting consequences from app interactions.

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. A set of "Safety Properties" (SPs) defines the proper operation of the entire system and may be either expressed explicitly, much like ORs, or implicitly through some code that checks for inter-operation of the subsystems. Both the ORs and SPs may involve time or time duration, and thus can be expressed explicitly in Linear Temporal Logic (LTL). The inability to simultaneously satisfy both SPs and ORs is considered a conflict. A simple way to handle conflict, considered in much of the work on the subject, is to block one of the conflicting actions when the conflict occurs. Other, more sophisticated ways include modifying the operational rules minimally to either avoid the conflict altogether or reduce the chances of its occurrence. The Section 5 we discuss three ways of approach this; namely, (a) avoidance based on the static analysis of ORs and SPs and possible modification to the ORs; (b) dynamic detection of conflicts and resolution by taking some action when they occur; and (c) anticipation of conflict dynamically but somewhat ahead of when they actually happen and handling them by changes to ORs.

2.2. Dependencies and Conflicts

A large-scale IoT system spanning the same physical area contains a plethora of sensors and actuators, plus a bunch of controllers and control loops. For example, an IBMS may use separate controllers with peer-to-peer relationships to manage energy and security functionality. A lower-level controller may interact with a higher-level controller on a more refined time granularity in other scenarios. For instance, the surveillance controller, which operates on a finer time scale, may continuously monitor and interact with people's movements and activities.

Given the possibility of overlapping operational rules, multiple controllers become interdependent. This occurs for various reasons, as discussed in detail in Abdullah, et al. [17]. Nonetheless, the primary ones are as follows: (a) various controllers striving to control the same actuator; (b) actions of multiple controllers that are dependent on the same sensors or parameters of the same shared environment.

A significant reason for dependencies, and hence conflicts, is the operation of multiple subsystems in the same shared physical space. The attributes of the shared space measured by the sensors (e.g., temperature, motion, luminance, etc.) provide a natural coupling between the subsystems since they may take actions based on these attributes, which may be conflicting. Note that the dependency here is on the measured attribute—it does not matter whether the measurement is performed by single sensor shared among the subsystems or by separate ones.

A somewhat different scenario is two different actuators (owned by different subsystems) taking actions that affect the same attribute (opening a window and HVAC control, both of which affect the temperature). The dependency may involve both actuator and sensor, such as closing the window affecting the sensing of smoke by the smoke detector.

The notion of "shared physical space", although intuitively clear, can be difficult to characterize precisely. Essentially, it depends on the physics of the situation, and even the notion of "same attribute" can be challenging to define. For example, the temperature or luminance in an adjoining room depends on many things, including whether the doors/windows are open, physical separation, precise locations of the actuators and sensors, etc. This can result in substantial complexity in determining whether the actions of two different controllers are interdependent and to what extent. Dependency, in particular, may occur in response to specific conditions and their duration. For example, the temperature in the adjacent room, say B, be sufficiently affected by the temperature in room A, only if the doors/windows connecting them are open for more than 15 min.

In addition to dependencies caused by shared space, there may be logical dependencies caused by some common element between the systems. For example, closing a water valve may impact nonshared and nonadjacent areas. Similarly, road congestion/closure at some spots may have substantially nonlocal impacts. These dependencies are related to multiple aspects, including the physical design/configuration of the system (e.g., water distribution network), physics (e.g., traffic flow), and human behavior (e.g., alternate routing).

In an extensive system, dependencies can often be caused by structural aspects, such as multiple devices of the same type using the same configuration or aggregation of various inputs before the controller sees it. Such structural commonalities induce dependencies in IoT system behavior, and hence potential conflicts.

The conflicts caused by the dependencies can take many forms, as described in Abdullah, et al. [17]. Given the discussion above, the conditions leading to a conflicting action could depend on many dynamic aspects of the system, and thus enumerating or anticipating all possible scenarios is difficult. Furthermore, even if a conflict scenario is known to be possible, it may not be possible to reasonably alter the operation so that it is always avoided, or it may be too expensive to do so.

2.3. Impact of Multiparty Environment

Smart environments are likely to sport multiple IoT subsystems, each with providing certain functionality or coverage area, and possibly designed/manufactured by a different company, and when deployed, operated/managed by yet another company. Take, for example, a smart building. It is likely to have a security/surveillance subsystem, energy management subsystem, and fire/safety management subsystem, each with very different functions, and thus, different controllers, which must be deployed, operated, and serviced by a different vendor. We will refer to these as different *parties*, and there could be many depending on the environment. In a smart city context, for example, it is possible that different parts of the city have different branded systems, possibly operated by different parties.

As stated above, in a large single-party IoT system, conflicts occur due to the difficulty of anticipating or statically avoiding it. Nevertheless, it is possible to check at run time the actual or potential conflicts provided that the controller has access to all the information. In a multiparty environment, however, there may be no global availability of information due to the autonomy of each party or simply the infeasibility of having a consistent global state in a large system. In particular, each party may have its own *Operational Rules* (ORs) that are not known to other parties. A party may not know about what sensors/actuators another party has or have any access to them. If the parties are entirely in the dark about others, no conflict detection or resolution is possible. On the other hand, unrestricted access across parties may be difficult to provision and provide, privacy/independence considerations, and potential security risks.

2.4. An Illustrative Example

Figure 1 illustrates some of these aspects of an IoT-based system. We have two IoT subsystems, each with its controller, one or more actuators, and some sensors. We assume that the sensors can be shared freely among subsystems, but each actuator is owned by precisely one subsystem. Subsystem 1 is deployed in two adjacent areas (denoted by orange and blue clouds) that overlap physically or functionally. An example of physical overlap is a connecting corridor containing sensors 2, 3, and an example of functional overlap would be lighting in either area that affects sensors 2, 3. Controller 1 receives readings from four sensors and operates actuator 1 based on sensors 1, 2, 3, and actuator 2 based on sensors 3, 4. Subsystem 2 is deployed only in the second area, receives readings from sensors 3, 4, 5, and operates only actuator 3 based on these readings. These relationships are shown using solid lines.

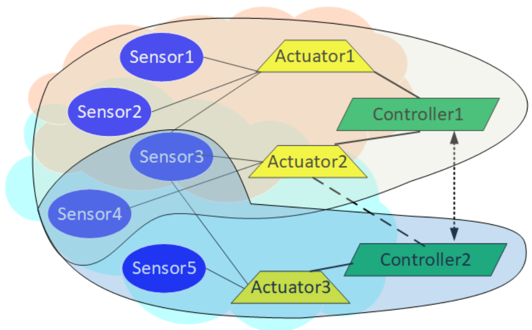

Figure 1. Internet of Things core model.

This system shows many dependency aspects of the problem. Sensors 2, 3 operate in the shared region and are thus affected by the environmental changes in either area. Of these, sensor 3 is *directly* used by (and hence affected by) Actuator 2, but Sensor 2 will also be affected indirectly. Depending on the rules, a change in sensor 3 may activate Actuators 1 and 2 either redundantly or oppositely. Actuator 2 is located in the overlapping region, and thus its action can have an indirect impact on the operation of controller 2 even though it is controlled by controller 1.

Interdependencies between different *operational policies* can lead to *conflicts* that could complicate the operation of an entire system or result in a compromise by attackers described in Abdullah, et al. [17]. These dependencies, however, can be beneficial in that they can be used to detect anomalies and malfunctioning devices (e.g., a temperature sensor showing a very different reading than others in a shared area). In either case, a set of safety properties) (SPs) could be defined to ensure that the system does not end in an undesirable state.

In the above, we assumed that every actuator is owned by exactly one subsystem. This corresponds to the typical deployments of the subsystems but is inflexible. For example, Actuator 2 (window open/close) should be operable by the HVAC system under normal (nonfire) circumstances. Thus, controller 2 should be able to request the operation of Actuator 2 (as shown by the dotted line), although this is via Controller 1. Allowing such conditional access and ensuring conflict-free operation is essential for the flexible coexistence of multiple subsystems but was not considered in prior work on the subject.

IoT deployments continue to evolve in terms of physical aspects (for example, new devices are added, old ones upgraded or retired) and the operational environment (new or different vulnerabilities manifest themselves). To comprehend the various forms of conflict, their consequences, and their mitigation, a collective static and dynamic analysis of subsystems is required. Additionally, this must be accomplished in a time-sensitive environment where operational accuracy, the occurrence of conflicts, and their resolution are all crucial components.

3. Literature Analysis of Conflict Characterization

Informally, a conflict can be defined as any undesirable situation or could lead to unpleasant consequences. However, formalizing such notions is nontrivial, and the papers on the subject are defined in many different ways. In this section, we discuss various types of characterizations. The next section then discusses details of detecting conflicts defined in various ways. Figure 2 provides a tree diagram of our discussion on the Conflict Detection and Resolution survey.

The conflicts can be mainly classified into three types: (1) *Rule-Based conflicts*, (2) *Application-Based conflicts*, and (3) *Ontology-Based conflicts*. (1) *Rule-based conflicts* occur when the automation rules or policies conflict with each other wanting to perform an opposite or otherwise undesirable action on the same type of device. The definition and detection of rule-based conflicts can be further classified as follows:

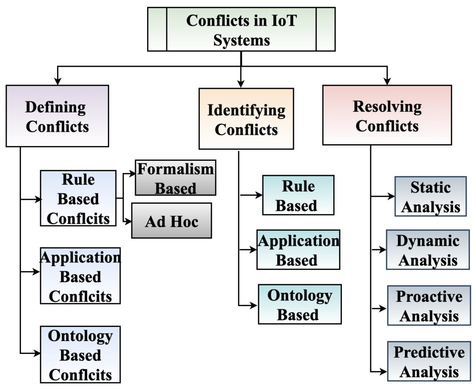

Figure 2. Overview of literature survey on conflict detection and resolution methods.

(a) Formalism-Based: here, the notion of conflict is introduced explicitly by defining a set of *Safety Properties* (SP) or a *Policy*. SPs are expressed in the same form as the rules but specify what must happen in a given situation to prevent undesirable behavior. Thus, a conflict is entirely defined in terms of logical conflicts (or contradictions) between the SPs and the rules. Such an approach makes the conflicts explicit. However, it requires that SPs be provided to prevent all undesirable behavior since any pair of actions that are not prohibited via an SP would be considered valid and nonconflicting in this model.

(b) Ad Hoc: here, the conflicts are defined by flagging certain behavior, expressed as a property of a set of rules operating together. Different researchers identified different types of situations and named them differently as conflicts. As with the formalism-based conflict definition, any behavior that is not explicitly flagged would be considered acceptable.

(2) *Application-Based conflicts* that occur when multiple IoT apps try to access the same device or have different/opposite effects on the same device or the environment.

(3) *Ontology-Based conflicts* that occur when IoT services are described using ontology and various types of conflicts are identified based on functional and non-functional characteristics of these IoT services.

3.1. Rule-Based Conflicts

3.1.1. Formalism-Based Rule Conflicts

Abdullah, et al. [17] proposed a formal approach for defining the (SPs) of an IoT system, and they described conflicts as a violation of these SPs by the system's ruleset. Their approach is quite different; rather than classifying conflicts, they propose a formal approach that specifies and categorizes the safety properties into several types. In particular, we may have scenarios where: (a) two different controllers trigger the same actuator at the same time; (b) two controllers affect the same attribute (e.g., temperature) in opposite ways; (c) two different controllers control parameters that in turn affect a common actuator; (d) two different controllers cause repeated opposing actions on the same actuator (e.g., repeated on/off).

Another piece of work is performed in the context of static and dynamic conflicts in Pavana, et al. [18]. The authors identified two distinct types of conflicts that can occur in a static or dynamic environment and developed an intelligent rule perturbation model to mitigate each type of conflict: (1) direct conflict that occurs for some action A, if there are two rules $C_1 \implies A$, and $C_2 \implies \neg A$, and both conditions C_1 and C_2 are simultaneously true; (2) extended conflicts that occur when a safety property is violated. 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.

Hsu et al. [19] modeled the interaction between devices and rules as an FSM in which the device statuses and automation rules correspond to the states and transitions, respectively. The FSM is verified against a set of predefined security policies, and violation of any security policy is identified as vulnerabilities among automation rules. Ma et al., in [20] further specifies safety requirements for a city and identifies the conflicts among smart city services based on the violation of the safety requirements.

Liu et al. [21] and Celik et al. [22] define conflicts as a violation of a policy. They consider conflicts in mainly two categories of policies: (1) user-defined policies, e.g., "if a user is away, then user-away-mode on", and (2) application-specific policies, e.g., "if the temperature is low, then turn on the fan". The conflicts are then classified into *racing events* and *cyclic events*. *Racing events* are defined as two or more events that are triggered concurrently with conflicting actions. In contrast, *Cyclic events* is defined as two or more events defined by a set of conditions and actions that continuously trigger one another.

3.1.2. Ad Hoc Rule Conflicts

In their work, Shah et al. [23] define incompleteness conflicts which refers to incomplete rules that do not cover all possible sensor values. One such rule is "turn on the sprinkler if the soil moisture content is less than 20%". According to this rule, the system activates the sprinkler when the soil moisture levels fall below 20%, and moisture levels continue to rise until they reach a point where additional water is detrimental. Hence this rule is insufficient because it omits a "turn off" action when the soil moisture content exceeds 20%. Generally, a set of IoT rules is incomplete if they do not have an "anti-action" defined. For example, consider two rules: Rule 1: if the temperature is greater than 100F, turn on the air-conditioner; Rule 2: if the temperature is less than 80F, turn off the air-conditioner. In this ruleset, Rule1 forms a trigger rule, and Rule2 is an antiaction rule that turns off the air-conditioner.

Chaki et al. [24], Abusafia et al. [25], and Lakhdari et al. [26] discusses various types of conflicts of available IoT services. Chaki et al. [24] proposed a novel IoT conflict model that encompasses both functional and nonfunctional properties of IoT services in a multiresident environment where residents may have different habits, resulting in IoT service conflicts. The main categories of conflicts include: (1) *Functional Conflict*, which occurs when multiple residents have varying state requirements for a service's functional property at the same time—for instance, simultaneously turning on and off a service. (2) *Nonfunctional Conflict*, which occurs primarily when residents have conflicting Quality of Service (QoS) preferences for the same IoT service. Nonfunctional conflict is further classified into conflict over resource capacity, qualitative nonfunctional conflict, and quantitative nonfunctional conflict. When there is insufficient capacity to use an IoT service by more than one resident, a resource capacity conflict occurs. Qualitative nonfunctional conflict is distinct from quantitative nonfunctional conflict in that it happens when residents have varying nominal over numerical QoS preferences for the same service concurrently. (3) Service Impact Conflict occurs when multiple residents prefer to access multiple IoT services that interact with one another directly or indirectly. However, this work places a greater emphasis on "opposite conflicts" than other types of conflict and lacks dynamic definitions of conflict types.

Huang et al. [27], Permula et al. [28], Shah et al. [23] and Oh et al. [29] defined conflicting situations in Event Trigger Action (ECA) rules, a popular mechanism to execute a rule in any IoT system. In an ECA rule-based format, events change the state of the IoT system, and if this change triggers the condition, the actuator performs the necessary actions. The two primary types of ECA rule conflicts are *direct conflicts*, which occur when multiple ECA rules compete to invoke shared IoT services at the same time, and *indirect conflicts* occur when multiple IoT services interfere with one another either directly or indirectly as a result of their impact on a shared environment entity. Another type of conflict that frequently occurs in ECA-rule-based architectures is a *static conflict*, in which two services are specified to perform contradictory actions on the same entity, and a *dynamic conflict* occurs during run time as a result of some policy actions being triggered indirectly in nondeterministic sequences. However, most of these works do not address efficient detection of runtime conflicts.

Huang et al. [27] is primarily concerned with developing a *conflict taxonomy* for defining various types of conflicts using ECA rules through the creation of a context-based knowledge graph that serves as a knowledge base for conflict detection. They defined four categories of conflicts: (1) *function–function conflict* occurs when two ECA rules attempt to invoke the same service in fundamentally opposite ways, for example, a rule indicating that windows should be closed and another rule attempting to open the windows. (2) *Cumulative-environment-impact conflict* occurs when two IoT services have an effect on a shared environmental property at the same time. For instance, fully opening the blinds on a sunny day and simultaneously turning on the heater may increase room temperature. (3) *Transitive-environment-impact conflict* conflict occurs when an IoT service's impact on the environment prompts the execution of another IoT service. (4) *Opposite-environment conflict* occurs when two ECA rules modify the shared environment in the opposite direction. For instance, when the air conditioner is cooling the room and the window is opened, the room temperature may be affected.

Sun et al. [30] decomposes complex rules into multiple basic rules and develops a formal 78 to analyze relations among these basic rules. The authors came up with a fine-grained division of 11 rule relations such as rules involving similar triggers, actions, prestate, poststate, etc. Based on the rule relations, they classify the rule conflicts into *(1) shadow conflict*, *(2) execution conflict*, *(3) environmental mutual conflict*, *(4) direct dependence conflict*, and *(5) indirect dependence conflict*.

Shadow conflict occurs when two rules involve the same actuators, and the trigger condition of one rule is contained in the trigger condition of the other. For example, with Rule1: "brightness < 400lux $\longrightarrow$ open curtain" and Rule2: "brightness < 600lux $\longrightarrow$ open curtain", execution conflict occurs when the trigger condition of one rule is contained in another and actions of both the rules are contrary. For example, with Rule1: "temperature < 45F $\longrightarrow$ turn on heater", and Rule2: "temperature < 65F $\longrightarrow$ turn on the heater". Environmental mutual conflicts occur when two concurrent rules are executed and a change in an environment entity is inconsistent as a result of the execution of one rule. For example, Rule1: "temperature < 30C $\longrightarrow$ turn on air-conditioner and keep temperature below 28C", and Rule2: "temperature < 25C $\longrightarrow$ turn on heater and keep temperature above 28C". Direct dependence conflicts occur when there is direct dependence between two rules and the trigger condition of one rule is in the state after execution of another rule and vice versa. For example, Rule 1: "turn on air-conditioner $\longrightarrow$ close window", and Rule 2: "close window $\longrightarrow$ turn on the air-conditioner". Finally, indirect dependence conflicts hold for multiple rules, and triggering a condition of one rule's state after execution of another rule, and so on. For example, Rule 1: "turn on air-purifier $\longrightarrow$ close window", Rule 2: "close window $\longrightarrow$ close window blinds", and Rule 3: "close window blinds $\longrightarrow$ turn on air-purifier".

3.2. Application-Based Conflicts

Motaz et al. [31] identified and classified conflicts based on two types: (1) application-level conflict occurs when multiple applications compete for the same resource, such as building administration applications attempting to control the temperature of a room, and (2) policy-level conflict occurs when automation rules or contextual policies conflict.

Nakamura et al. [32], Leelaprute et al. [33,34], and Igaki et al. [35] model the feature interactions and the resulting conflicts in HNS. An HNS consists of networked home appliances such as TVs, air-conditioners, lights, refrigerators, etc. There are two distinct types of feature interactions proposed in these papers: *appliance interaction* and *environment interaction*. The HNS is modeled using an *object-oriented* approach where each appliance is an *object* consisting of *properties* and *methods*. The properties describe the appliance's internal states, whereas the methods abstract the appliance's features. The appliance's properties may be referenced or updated during a method's execution. Hence, an HNS system is a set of appliances and environment objects. Conflicts are defined as interactions between the appliance and the objects or methods in the surrounding environment. The appliance interaction is in direct conflict if it occurs between two methods of the same appliance device, and it is in indirect conflict if it occurs between two methods of different appliances that update a common environmental object.

To detect inter-app conflicts, it is critical to have a precise definition of conflicts between multiple apps. Li et al. [36] define inter-app conflicts in two broad categories: *strong* and *weak* conflicts. When multiple apps run concurrently, a strong conflict can occur when some of an app's actions are disabled due to interaction with other apps; a weak conflict occurs when the apps control an actuator differently but no app's actions are disabled.

3.3. Ontology Based Conflicts

Chaki et al. [24] and Camacho et al. [37] describe IoT services in a multi-household smart home using ontology. They defined a conflict ontology that establishes various types of service conflicts resulting from the inconsistency of services due to multiple users accessing them with varying preferences. This conflict ontology primarily defines conflicts in whether they occur within a single service or across multiple services.

4. Literature Analysis of Conflict Identification Methods

As discussed in the last section, conflicts can be defined from many different perspectives, and their impact can also be varied, ranging from minor inconsistencies to major mishaps. Timely detection of conflicts is thus essential to its resolution, which itself may range from simply ignoring it to entirely blocking certain actions (or, alternatively, initiating a counteracting action). In this section, we discuss different ways of representing the operation of the IoT system and corresponding conflict detection mechanisms. Figure 2 provides an overview of our discussion on conflict detection methods.

4.1. Operational and Conflict Representations

The conflict identification mechanisms can again be classified into three main categories, "Rule-Based", "Ontology-Based", and "Application-Based", which we will discuss next.

In a rule-based system, the IoT services are expressed as operational rules defined in the form trigger→action. An ontology can be defined as a specification of the conceptualization of a domain. Ontology-based frameworks enable the representation of knowledge through the use of well-defined semantic and syntactic rules. Ontologies are collections of domain knowledge encoded using axioms, natural language labels, synonyms, and definitions. They facilitate the creation of reusable entities and relations among devices, events, and activities in an IoT system. In an ontology-based approach, a conflict is defined as an interference with the user's preferences or when postconditions for initiating an event conflict with preconditions of a happening event. Application-based work is concerned with resolving inter-application conflicts; the presence of multiple apps controlling the same actuator or device results in potentially undesirable interactions. For instance, in a smart home, the owner has two apps: one that sounds an alarm when smoke is detected, and another that unlocks the doors. Additionally, the same owner installed an app that automatically locks the door when the homeowner leaves the house. Even after these apps were safely installed, they interact unexpectedly, resulting in inter-app conflicts.

4.2. Rule-Based Tools

Rule-based tools used many representations, with the most popular one being IFTTT ([38]). It is a web-based application that specifies trigger-action using "If This Then That" (IFTTT). It is one of the simplest ways to configure system behavior by identifying triggers, such as "if the user at not home", then "turn-on security cameras". The IoT services developed with IFTTT are referred to as Event-Condition-Action (ECA) rules. ECA rules are a promising approach due to their compact and intuitive structure, which connects dynamic events and or conditions (triggers) directly with the expected reactions (actions) without requiring the use of complex programming structures.

Several other tools are also available to manage and configure an IoT system behavior. Atooma [39], dubbed as "A Touch Of Magic", which uses if-then construct for defining automation rules. It can be integrated with mobile applications, web services, and external devices via a single interface. Atooma enables the selection of up to five conditions and their associated actions and the recognition and recommendation of frequently occurring situations. Tasker [40] is an android application that executes tasks (sets of actions) in user-defined profiles or widgets for home screen, based on the context (application, time, date, location, event, gesture, etc.). This straightforward concept significantly expands the control and capabilities of the Android device without the need for 'root' or a unique home screen. Bipio [41] is a Graph API that enables users to create customized workflows and to concatenate multiple services via the provided application logic, which is based on the Remote Procedure Call (RPC) protocol. As with Atooma, Bipio does not require any programming knowledge; instead, each node in the user graph is assigned responsibility for executing a unit of work to transform messages, integrating different web services, or creating other types of web applications via RPC protocol.

WigWag ([42]) is an open-source mobile app that provides automation for ambient intelligence allowing for instant control of connected devices in a home or a building. Zipato ([43]) is a rule-based home management system for IoT service automation. It requires a dedicated gateway, called Zipabox, to which the system's numerous devices (sensors and actuators) are connected. The operational rules are created using Zipazle, a visual programming environment based on the MIT Scratch project accessible via the web on smartphones, tablets, and computers.

4.2.1. Rule-Based Conflict Identification Methods

Huang et al. [27] propose a generic knowledge graph model to represent the relationships between ECA rules defining IoT services and environmental properties. The proposed algorithms identify overlapping event pairs in the graph and the resulting direct and indirect conflicts. The algorithm performs well on a small number of real-world and synthetic datasets, but the authors only evaluated their proposed work statically on predefined rules. The paper does not demonstrate the impact of rules in a real-time setting that could affect the environment inside a smart home.

Perumal et al. [28] presents a framework for detecting and resolving conflicts via a weighted-priority scheduling algorithm. The proposed work captures and prioritizes events generated by heterogeneous systems in a smart home environment and utilizes a weighted scheduling algorithm to detect and resolve conflicts. If two ECA rules are in direct or indirect conflict, the event associated with the highest-priority rule condition is scheduled and triggered first. The work assigns a predefined priority to events and executes them accordingly. Although simple, choosing one of the conflicting events based on a fixed priority is not satisfactory since the appropriate action often does depend on the overall context.

Oh et al. [29] developed a framework for detecting ECA mashup service conflicts in a smart home environment. The proposed mechanism defines a context descriptor for each mashup service instance that includes the following properties: *Instance* IoT service, which performs actions when the mashup service is processed; *Context* condition, which is impacted by the instance; and *Direction*, i.e., direction of context change. As an illustration, consider the use of an air purifier as a mashup service for the home environment. When it is activated, the indoor air quality improves. “Air purifier” is an instance, “indoor air condition” is the context, and “improves” is the direction in this example. The service conflicts are then identified by tracking context descriptors associated with the mashup service chains. The primary disadvantage of the proposed work is that the service conflicts are detected statically via context descriptors; the proposed work does not address dynamic conflict detection.

Sun et al. [30,44] proposed a conflict detection module that identifies predefined relationships between rules and uses these relationships to classify rule conflicts. A rule database is defined in an XML format and includes a rule table, a location tree, and an authority tree to filter out useless information to further improve detection efficiency. The work proposed achieves higher efficiency in comparison to that of other similar works. However, the work is based on static rule checking of simple rules and is incapable of detecting conflicts between complex policies. Moreover, the rule parsing technique frequently produces inefficient results in recognizing the actual conflicts.

Several works were performed to detect conflicts in trigger-action programs, e.g., Luo et al. [45], Maternaghan et al. [46], Sun et al. [30,44]. Luo et al. [45] proposed a rule-verification mechanism based on a probabilistic analysis to determine the possibility of rule conflicts and anomalies. There are many studies on detecting conflicts in an HNS, as presented in Nakamura et al. [32], Leelaprute et al. [33,34], and Igaki et al. [35], where each appliance was modeled as an object consisting of *properties* and *methods*. The behavior of an appliance is defined by state transition rules that include a *pre- and postcondition*. Feature interactions are defined as conflicts between integrated services. These works consider feature interaction conditions in Linear Temporal Logic (LTL) formulas and validate them against HNS services. They consider both offline and online feature interactions. Leelaprute et al. [34] proposed a classification of these feature interactions and resolution schemes for each type of interaction.

4.2.2. Formal Methods for Conflict Identification

Newcomb et al. [47] and Bal et al. [48] were based on a core calculus for IoT Automation that generalizes ECA rules for home automation apps. Newcomb et al. whose work was named IOTA [47], involved the development of the first calculus for the domain of home automation. The evaluation part of that work includes conflict detection as one of the possible uses of the calculus by developing an analysis for determining when an event can trigger two conflicting actions based on the IOTA core formalism and a study for determining the root cause of an event's (non)occurrence. Their conflict detection framework is based on a model checker that takes all the rules and, upon detecting a conflict, generates traces that detail sample execution, including the environment's initial state, the event that triggered the conflict, and the rules involved.

Other calculus similar to IOTA are performed in Sen et al. [49] named as dT-calculus and in Xie et al. [50] called mCWQ . The dT-calculus is a formal method for describing a distributed mobile real-time IoT system. The authors go over other calculi in this paper, such as Time pi-calculus and d-Calculus. Time pi-calculus can specify timing properties; however, time pi-calculus does not have direct specifications of action execution time and process mobility, such as ready time, timeouts, and deadline. d-Calculus allows the only simple type of temporal requirements to be specified, for instance, a temporal constraint on the minimum and maximum values. mCWQ is a mobility calculus that demonstrates how to capture the feature of node mobility and improve communication quality.

Shehata et al. [51,52] propose a semiformal model, IRIS (Identifying Requirements Interactions using Semi-formal methods) for defining and detecting policy interactions/conflicts in intelligent homes. The authors introduced an interaction taxonomy that examined interactions between policy features other than telecommunications features in various domains. A run-time module for detecting and resolving policy conflicts is defined based on simulation techniques. IRIS's conflict detection work considered negative impact conflicts, in which one attribute or feature hurts another, and override-conflicts, in which one attribute or feature overrides and cancels out the other. The disadvantage of IRIS is that it does not account for rule dependency and thus cannot detect direct and indirect dependence conflicts. Abdullah et al. [17] proposes a formal model approach for detecting conflicts within the defined ruleset of the IoT system that violate the safety properties defined in the system. The proposed work detects conflicts immediately after an event occurs, which may trigger an action or a series of actions that result in conflicts, and also demonstrates how conflicts can result in additional actuations, which eventually result in increased energy consumption.

Carreira et al. [53] detects conflicts as changes in the environment state that result in an undesirable application or user context, as expressed by a Constraint Satisfaction Problem (CSP). Based on the proposed conflict taxonomy and a CSP, the authors proposed a framework where the system context is represented by a set of variables and constraints. The system for detecting and resolving conflicts consists of four major modules: a context querier, a CSP solver, a solution translator, and a decision module. The context querier module receives context information and any number of context-related queries, all of which are described as environmental conditions. It then combines the context queries as additional conjuncts and converts them to a constraint system for input to the CSP solver module. The solution translator will receive the results from the CSP solver and convert them to the environment model's output format. Finally, the decision module analyzes the data to determine: (i) whether there is a conflict; (ii) whether the conflict is solvable or not; and (iii) whether the conflict is solvable through the environment or occupant modification, or a combination of the two.

4.3. *Application-Based Tools*

The application-based IoT automation platforms allow developers to write apps that implement functionality on smart devices in any domain. There are several platforms such as Android Things from Google [54], Samsung Smartthings [55] and SmartApps, Apple's Homekit [56] and the open-source OpenHAB [57]. These platforms enable interoperability of devices from various vendors via a local gateway (e.g., a hub or a base station) or cloud back-end servers. Third-party developers can create software applications to enable smart device control. For instance, an application may monitor the status of one device (e.g., a motion sensor) and initiate specific actions on another device (e.g., turn on the light) in response to receiving notification of a particular event (e.g., human activity). The framework's extensibility attracts many device manufacturers and application developers to the ecosystem.

Android Things is an IoT platform by Google that lets users operate IoT-connected devices using Android APIs combined with the Google Cloud Platform. Samsung Smartthings provides software developers with an abstraction of a variety of lower-layer smart devices, decoupling the development of software programs (i.e., SmartApps) from the manufacture of the smart devices. As a result, the SmartThings platform fosters a thriving software ecosystem, encouraging third-party developers to expand the breadth of home automation capabilities. SmartThings currently supports 133 device types and 181 SmartApps, according to the official GitHub repository. OpenHAB consists of 103 rules managing devices such as lights, window shades, heater, cooler, motion detectors, temperature, and humidity sensors, time and date, and so on. The rules contain an event handler and an action block. Apple's HomeKit is one of the best ecosystems for setting up a smart home with the Apple Home app, which serves as the hub of Apple's smart home and provides a centralized location to view and control a smart home.

Application-Based Conflict Identification Methods

IA-GRAPH developed in Li et al. [36] and Shen et al. [58] studies inter-app conflicts in a smart home domain and adopts an approach in which an app's transitions are represented as SMT formulas. Conflicts between multiple apps are then detected using an SMT solver. The advantage of using an SMT solver is that it enables the generation of model representations that accurately capture the interactions of device controls in an application's source code. However, the approach disregards complex application logic contained in condition statements with multiple time and threshold values, instead considers only simple logic which involves changing the state of a device with a binary value, i.e., on or off.

There exist several works on detecting and resolving conflicts across the IoT services provided by multiple applications in the context of smart homes [21,22,36,59–64] and in the context of smart cities [20,65–67]. Among these [59,62,65,68] are the examples of applications that resolve conflicts by assigning priority to different services based on the domain or administrator's understanding of each service, among others.

Lie et al. [21] propose a framework inspired by the conflict detector work Celik et al. [22], which is based on actuation graphs. Actuation graphs provide a polymorphic abstraction of IoT actuators and sensors, which is then used to formulate remedial actions for a given IoT policy conflict. This framework is evaluated on a set of SmartThings apps in which sensor and actuator states are binary, i.e., on or off, and the set does not take into account any thresholds associated with the sensor or actuator state. The original framework Celik et al. [22] considers conflict detector as a directed graph where each node represents a collection of device (or module) states, and each directed edge represents a trigger-action relationship. Further, a node may contain a cascade of conditions (actions) rather than just one module state. A conflict occurs when a set of compatible conditions (i.e., conditions that can be satisfied concurrently) result in mutually exclusive states of the same device. However, the framework is unable to deal with policies that represent complex interactions between smart devices and users.

4.4. Ontology-Based Tools

Ontologies are an integral part of Semantic Web stack technologies as shown in Berners et al. [69] for knowledge representation. The majority of IoT system ontologies are formalized using Ontology Web Language (OWL) developed by Grau et al. [70], which is a language based on description logic for expressing correspondences between ontologies. Description logics enable the formal, machine-readable description of various entities and their relationships within a domain. OWL includes explicit semantics that defines how OWL statements constrain the world they are interpreted. Ontology Web Language has been a W3C Recommendation since February 2004. A contextualized OWL (C-OWL) was proposed in Bouquet et al. [71], in which an extension of OWL was mainly defined to express mappings between heterogeneous ontologies.

Ontologies can be combined with automatic reasoning and inferring. For example, Semantic Web Rule Language (SWRL) developed in Horrocks et al. [72] extends OWL to include an explicit notion of a rule expressed in first-order Horn clauses. Rules written using SWRL can be interpreted as correspondences between ontologies, mainly when the head and body contain elements from distinct ontologies. SWRL rules have the advantage over genuine OWL of being well-identified as rules and are easier to manipulate as an alignment format than OWL.

Ontology-Based Conflict Identification Methods

Shah et al. [23] proposed a mechanism for detecting conflicts based on ontology using incomplete rules with a three-part scheme for resolving the conflicts. The first step is rule decomposition, which considers conjunctive clauses and their corresponding disjunctive normal forms to minimize rule conflicts. Following that, rule relationships are established, and finally, rule conflict incompleteness is determined using the relationships. Additionally, this method incorporates rule integrity to ensure that the sensor range of undefined actions is not exceeded. Experiments demonstrate that this method improves the rate of missed detection. Chaki et al. [24] designed a conflict ontology model that represents different types of conflicts. A hybrid conflict detection algorithm is proposed by combining both knowledge- and data-driven approaches to detect conflict among IoT services in multiresident smart homes. Camacho et al. [37] propose a conflict detection and resolution framework that performs context analysis based on an ontology that formally represents environment's conditions.

4.5. Static vs. Dynamic Conflict Detection Methods

Celik et al. [61,73] developed a static analysis tool for tracking sensitive data flows as well as finding protection and safety issues in an IoT application. The developed system converts the source code of an IoT application (SmartThings app) to an intermediate representation, extracts a state model from this intermediate representation, and performs model checking on IoT applications to identify property violations, which indicate whether or not an IoT application adheres to predefined safety and security properties.

Ma et al. [74] proposed topology of conflicts that can be detected at run-time. They also proposed a watchdog architecture for detecting and resolving rule conflicts in the context of smart cities based on simulation. Watchdog identifies various characteristics of smart city services that contribute to potential conflicts in smart cities, including uncertainty, real-time, dynamic behavior of services, and spatio-temporal constraints. Along with conflict detection and resolution, Watchdog's extension work, CityGuard in Ma et al. [20] specifies additional safety requirements for a smart city. Most of these works presuppose prior knowledge of the system's components and the rules governing their evolution. They do not propose conflict resolution strategies that are both local and reusable. On the contrary, they employ globally scalable identification and resolution mechanisms.

5. Literature Analysis on Conflict Analysis and Resolution Strategy

5.1. Classification of Conflict Analysis and Resolution

There are two basic ways of handling the conflicts: (a) *Avoidance* where potential conflicts are identified via static analysis (i.e., at design or initial configuration time) and resolved by changing the operational policy logic of the system, and (b) *Resolution*, where the conflicts are detected at run-time and actions are taken to diffuse them. Figure 2 provides an overview of our discussion of conflict analysis and resolution methods.

Static analysis tools build a semantic model of the software at compile time without executing it and then check various properties of the model. Static conflict resolution is performed offline, and it is necessary to identify all conflict types that must be detected, including conflicts that are evident from the policy specification and that are not evident from policy specification but arise as a result of policy dependencies. Because most subsystems are deployed incrementally, static checking cannot completely eliminate all conflicts because of the following factors: (a) the subsystem's operation and interaction with others continue to evolve as devices are added, removed, or upgraded; (b) static checking must consider all possible interactions, which is difficult and often impossible; and (c) many statically identified (potential) conflicts may require highly unlikely or even unrealizable scenarios, and thus trying to avoid their many perfectly reasonable sets of actions would thereby handicap the system substantially. Finally, in a multiparty environment, it is not even possible to do a static analysis unless all parties are willing to share their detailed operational semantics and agree to making the identified changes.

Dynamic resolution can operate in multiple ways, the extreme case being where the conflicts are checked dynamically for each action, and if a conflict situation is recognized, it is resolved by taking some action, often by simply blocking one of the conflicting actions. There are several advantages of such a method over the static analysis: (a) the conflict detection is simple in that we only need to check the relevant *Safety Properties* (SPs) before taking action, and block the action (or initiate a counter-action) so that the conflict is avoided; (b) all conflicts, in so far as they are codified in the SPs, can be checked without having to anticipate/enumerate scenarios in advance; and (c) a corrective measure (such as blocking/disabling an action) is needed only when the situation demands it, not as a result of a potential conflict.

However, detection of conflicts when they are experienced and a reactive resolution can be undesirable and sometimes dangerous in cyber–physical systems since we effectively wait until a problem happens, rather than anticipating the problem and addressing it (if possible). As explained next, there are two related approaches to do this, which we term as proactive and predictive.

5.2. Proactive Resolution

By proactive, we mean that the system looks ahead by specified time duration and examines the *likely events based on the current conditions*. If possible, these conflicts can be resolved proactively by modifying the involved entities. The proactive model needs to identify the events that can occur in the near future based on the context, physics of the situation, and possibly the history of previous events.

Apart from higher complexity, the proactive method necessarily introduces uncertainty: the predicted/anticipated conflict may not actually occur in which case the corrective action becomes unnecessary. There is an obvious (but difficult to characterize) trade-off between how far in advance we analyze the conflicts (say, time Δt) and the corresponding false positive and false negative rates. Yet another difficulty is the characterization of *likely events based on current conditions* during the time Δt. If Δt is sufficiently small, it is reasonable to assume that no new events will occur, and the dynamics of the system will be governed by what is currently ongoing. For example, if the smoke density is currently increasing due to something burning, we assume that this will continue. Similarly, if the car is getting closer to the one ahead, this will continue. Obviously, such assumptions become increasingly untenable as Δt increases, but there is also uncertainty about what actions may or may not occur as a result, and these would also need to be predicted. Furthermore, if a new event or action does occur, we need to somehow transition from the current evaluation to a new one that takes the new situation in the account (e.g., start over entirely or determine what aspects of the prediction may have changed).

A formal analysis of the system is essential to detect existing conflicts, anticipate potential conflicts in the future, or determine actions to resolve or avoid the conflict. The basic analysis techniques needed for this are much broader and can also be used to check other properties of the IoT system. Other than operation, this also includes access-related properties, provided that the formulation also includes allowed or disallowed accesses. More details on the proactive resolution strategy can be found in Section 6.

5.3. Predictive Resolution

By predictive, we mean a mechanism that observes the behavior of the system over time and perhaps over multiple occurrences of a similar sequence of events and actions and learns from them. The key distinction between this and proactive is that: (a) predictive is not necessarily tied to short time-window behavior that we exploit in the proactive method; and (b) predictive-based decisions are based on multiple occurrences of similar situations, rather than on the short-term physics of the situation.

To the best of our knowledge, there are no works that exploit prediction in this sense for conflict resolution purposes; however, many works attempt to predict user behavior in an IoT system. These and related mechanisms could potentially be integrated with conflict detection and resolution to provide this class of solutions. Coppers et al. [75] proposed a framework, FORTNIoT, for making intelligible future predictions by combining self-sustaining predictions (e.g., weather forecasts) and simulations of trigger-condition-action rules, to ascertain when these rules will trigger in the future and what state changes they will cause to connected, intelligent home entities; however, this study falls short of detecting conflicts during the prediction of future activities in a smart home. Numerous algorithms were developed to predict the user's behavior in a smart home. For instance, a user typically awakens at 8 a.m. and immediately operates the toaster on weekends. Any method for predicting user behaviors should mine the user's behavior from such operation records and return it to the smart home control center. If tomorrow is a weekend, the system will ask the user one day in advance whether they require assistance with using the toaster at 8 a.m. Liang et al. [76] proposed an algorithm for Unsupervised User Behavior Prediction (UUBP) that utilizes an Artificial Neural Network (ANN) to learn user behavior along with an innovative update strategy that integrates an Ebbinghaus forgetting factor. Further, the Forgetting Curve is proposed to eliminate the influence of infrequent and out-of-date operation records generated by the user. This can help to mitigate the impact of out-of-date records on the prediction process, resulting in more predictive behaviors that are more consistent with recent user behaviors. This work detects and resolves the conflicts in user behavior records that might trigger the same actuator simultaneously. Du et al. [77] developed a model based on a Long Short Term Memory (LSTM) network to predict Activity of Daily Living (ADL), or the next activity that may occur after the user's current activity. Wu et al. [78] presented a comprehensive survey of prediction algorithms proposed in the literature for smart environments. The algorithms presented predicts future events based on historical data drawn from sensors and smart devices to reduce the likelihood of malicious events occurring. This paper introduces the system models and data that are commonly used in smart prediction algorithms and discusses their features, strengths, and weaknesses in detail.

5.4. Static Analysis of IoT Systems

Static analysis techniques can be used to (1) validate that a collection of IoT apps adheres to identified safety, security, and functional properties (e.g., locking doors when the user is not at home); (2) identify whether specific properties or relationships that are considered sensitive can be inferred from the state information used for analytics (e.g., whether the user is at home or away or whether the door is locked); (3) identify misuse of an IoT app's permissions to access sensitive data. For instance, a smoke-alarm app may request permission to disable a security camera, even though the app does not require the permission to function; and (4) develop a provenance system that provides a complete history of device actions and events to identify an attack or misconfigurations.

Static analysis can be accomplished in two ways: (1) event-based analysis of IoT applications and (2) a more detailed flow analysis of the source code. The source code for an IoT application can be translated into a platform-independent structure composed of three types of common building blocks, namely, (1) *Permissions* that allow the app to access devices and user inputs used to implement the app's functionality; (2) *Events* that represent the relationship between sensor readings and actuators; and (3) *Call graphs* to represent the relationship between main methods and functions in the app.

Celik et al. [22,61], Ding et al. [64], and Nyugen et al. [63] performed a static analysis of smart applications to identify possible interactions that violate safety and security requirements on the Smartthings platform even though the individual apps operate safely. Additionally, Celik et al. [22,61], and Ding et al. [64], analyze the path and context-sensitivity of the application's event handlers by creating a distinct call graph for each entry point. Their work focuses on performing dependency analysis on the source code of an application to identify potential sources for numerical-valued attributes and then pruning sources based on path and context sensitivity to construct a state model of the app that includes its states and transitions. Further, a union state model is constructed that represents the cooperative behavior of apps and performs model checking to identify data leaks and violations of safety and security requirements.

IoTSAN in Nyugen et al. [63], VISCR in Nagendra et al. [79], and HomeGuard in Chi et al. [80] are all based on source code analysis of IoT applications, and thus take into account additional factors such as state explosion reduction and analysis of specific malicious input sequences. These tasks require translating the application's source code to perform model checking, which identifies interaction-level flaws by identifying events resulting in the system entering an unsafe state. These works, however, are incapable of detecting violations mediated through physical channels.

Many of these approaches do not track how data flows statically from IoT end-point devices through the Internet to a cloud layer (or vice versa). In this context, static program analysis can be highly beneficial to determine the taint of data propagating across different IoT layers (e.g., sensitive or user-controlled data). Taint analysis in Celik et al. [73], Fernandes et al. [81] and Bastys et al. [82], in particular, determined whether something from a source (e.g., methods for retrieving user input or sensitive data) flows into a sink (e.g., ways for sending data to the Internet or running Structured Query Language (SQL) queries) without being sanitized (e.g., encrypted or escaped). SAINT in Celik et al. [73] is a three-phase static taint analysis tool that (a) converts source code for IoT applications to an intermediate representation that encapsulates the app's life-cycle, including program entry points, user inputs, events, and actions; (b) identifies sensitive sources and sinks; and (c) performs static analysis to identify sensitive data flows.

FlowFence in Fernandes et al. [81] is a framework that proposes a method of restricting access to sensitive IoT data. The proposed solution enables developers to partition an application into two modules: the first module manages sensitive IoT data in a sandbox, while the second module coordinates the transmission of such sensitive data via integrity constraints. The validation of FlowFence in the consumer IoT realm demonstrated that confidential information is preserved with a minimal increase in overhead. Bastys et al. [82] reported three classes of URL-based taint analysis, based on URL markup, URL upload, and URL shortening in IoT apps, conducted an empirical study to classify sensitive sources and sinks in IFTTT control flows, and proposed an efficient method for taint analysis using the Java Script flow tool.

However, these approaches are limited in precision and the number of privacy policies enforced. For instance, SAINT and FlowFence have no way of knowing whether an app leaks sensitive data through the developer or user-defined strings. In addition, SAINT does not consider sensitive data leaks at run time, while FlowFence often over-estimates the data leaks. Moshin et al. [83] presented IoTSAT, a formal framework for security analysis based on device configurations, network topologies, user policies, and IoT-specific attack surface. However, this research does not consider either safety properties or address conflicts for smart environments.

To minimize the attack surface, it is necessary to understand the behavior or profile of IoT devices. Text data crawling and NLP (Natural Language Processing) are used to determine whether the device's behavior is consistent with the design goal or is performing some unauthorized activity. This rule extraction module automatically extracts behavioral rules from devices. The module is subdivided into two modules: one for interactive rule extraction and another for communication rule extraction. The interactive rule extraction module crawls the current user interface to obtain basic device information. Then, NLP tools extract rules governing device interaction from the device information.

Different from source code analysis of IoT apps, IoTMon presented in Ding et al. [64], SmartAuth developed in Tian et al. [84], HomoNit presented in Zhang et al. [85], WHYPER in Pandita et al. [86], and IoT-praetor developed in Wang et al. [87] present an approach that utilizes NLP techniques to identify behavior models from IoT apps for model checking or risk analysis. Such an NLP-based approach enables security analysis in the absence of source code. These studies infer whether IoT applications adhere to the original design objectives or engage in some unauthorized behavior. In mobile and IoT applications, NLP automatically extracts security-relevant information from the app's description, code, and annotations. The extracted semantics are compared to the program's tracked control and data flows to detect the application's misbehavior, which necessitates complex program analysis techniques.

IoTMon is a static analysis technique that utilizes text mining of app descriptions to discover common physical channels between IoT devices and possible physical interactions between devices. Additionally, IoTMon includes a risk assessment of each discovered inter-app interaction chain based on its physical influence. However, IoTMon does not consider the temporal aspect of physical interactions. Certain physical interactions between IoT devices can occur instantly. For example, turning on a light immediately activates an illuminance sensor. On the contrary, physical interactions occur gradually but continuously.

SmartAuth is a context-aware authorization mechanism that extracts security-relevant information from the code and description of an application and generates a user-friendly authorization interface, through which a user can specify which capabilities the application has access to on a per-device basis. SmartAuth incorporates code analysis and NLP on app descriptions to create a new policy enforcement mechanism compatible with existing home automation frameworks and enforced complex, context-sensitive security policies with minimal overhead. HoMonit used NLP to compare the activities of smart applications as determined by side-channel analysis of encrypted wireless traffic to their expected behavior as specified in their source code and discovered 60 misbehaving apps that engaged in event-spoofing attacks. However, these existing solutions either require modification of the platform itself or patching the apps.

All of these works are limited to the Samsung Smartthings platform. Android applications also make use of NLP techniques. WHYPER is the first work to use NLP techniques to bridge the gap between the semantics and behaviors of Android apps. It extracts semantic information from app descriptions and API documentation and then determines whether the descriptions justify the use of specific permissions. The goal of WHYPER is to determine whether the need for sensitive permissions is motivated in the application description. However, this work requires manual annotation of sentences describing the need for permissions.

5.5. Dynamic Analysis of IoT Systems

Dynamic analysis approaches shown in Pavana et al. [18], Fernandes et al. [81], Jia et al. [88], Wang et al. [89], and Babun et al. [90] do not have the drawbacks associated with static analysis. As stated in Ernst et al. [91], static analysis may result in over-approximations by generalizing all possible behaviors of an application from its source code, potentially resulting in false positives. For instance, an analysis tool in Celik et al. [73] detects a sensitive data leak prompted by a nonexecutable piece of code in an IoT application.

IoT-Watch presented in Babun et al. [90] is a run time analysis tool to identify privacy violations in smart home apps. IoT-Watch provides a straightforward interface through which users can specify their privacy preferences (e.g., location, device states, etc.) during the installation process. It then injects additional logic into the app's source code to collect information about the app during run time. Through NLP techniques, the collected information is used to classify the data sent out of the IoT app and inform the users when an app's leak matches with the privacy preference of a user. This work is based on an IoT privacy survey of human subjects that use different IoT devices and does not take into account other factors, for instance, ongoing events.

ContexIoT shown in Jia et al. [88] proposes a context-based permission system for the Samsung SmartThings IoT platform that identifies fine-grain context information and prompts users to make an access control decision at run time. They are the first to use a patching mechanism to collect run time data and pause the execution of SmartApps to segment the execution of a SmartApp into context collection and permission granting phases. However, this framework requires users to participate in decision-making at run time, which overshadows the benefit of home automation and, worse, may violate the SmartThings method execution time limits of 20 s if users do not respond in time.

ProvThings developed in Wang et al. [89] is a platform-centric logging framework capable of constructing data provenance graphs for all activities in an IoT system and utilizing them to deduce the cause of an abnormality. ProvThings is prototyped for Samsung SmartThings and is primarily used for forensics and not for proactive defense. By utilizing security-sensitive APIs, this framework keeps track of the system's provenance and uses it for forensic reconstruction. Fernandes et al. [81] proposed a framework, FlowFence, that splits a smart app source code into sensitive and nonsensitive modules and orchestrates the execution through opaque handlers. The proposed framework protects IoT data from leakage and misuse by using sandboxes and taint-tracking to enforce data flows between data sources and data sinks.

6. Proactive Conflict Resolution in IoT Systems

As discussed earlier, proactive conflict detection and corresponding resolution can ideally combine the best aspects of static and dynamic methods, i.e., accurate dynamic avoidance of conflicts that are likely to occur in the near future. Pavana et al. [18] devised an approach that not only detects conflicts proactively, but also resolves them by an automated loosening of rules rather than by simply blocking an action. We describe this next briefly; we are not aware of other proactive approaches.

6.1. Proactive Conflict Detection

To formalize conflict detection, we formalize the IoT system behavior in terms of (a) a set of *Operational Rules* (ORs) for each subsystem i, denoted $\mathcal{R}_i^{(o)}$, that moves the subsystem i from one valid state to another, and (b) a set of *Safety Properties* (SPs), denoted $\mathcal{S}$, that expresses the constraints on the behavior to avoid conflicts. An SP expresses the idea that "something (bad) should *not* happen" during the system execution. We assume that each subsystem by itself was designed to be conflict-free; therefore, SPs concern at least two subsystems. Both ORs and SPs can be expressed as a triplet *(precondition, trigger, postcondition)*. In the case of ORs, this means that if the IoT system is currently residing in a state characterized by the precondition, a trigger moves it to a state characterized by the postcondition. The trigger here could be an action (operation of an actuator) or an external event. Most (perhaps all) ORs concern an individual subsystem, although we do allow for cross-subsystem ORs for generality. An SP necessarily involves multiple subsystems (at least two). The trigger for an SP can again be an action or event, and the postcondition represents the *required state* for proper behavior. An SP could specify enforcement in that if the given (presumably anomalous) pre-condition holds, an action must be taken to correct the situation.

As a concrete example, an OR might specify to turn on the heater when the room temperature goes below 15C. This can be expressed as ((temp $>$ 25C & cooler = off, turnon_cooler), cooler = on). An example of SP is (cooler = on $\wedge$ heater=on, turnoff_heater, heater = off). Both of these can be easily converted into first-order logic expressions, e.g., ((temp $>$ 25 & cooler = off & turnon_cooler) $\implies$ cooler = on). As a result of this, conflict detection is reduced to the Boolean satisfiability problem, with the exception that we need appropriate theories (for example, theory of heat transfer), which requires the use of SMT solution approaches. Most popular tool for SMT is Z3 [92] tool. Along with theories encoding the physics of various processes, we need to incorporate time and temporal concepts directly into a model of the IoT system operations.

Linear Temporal Logic (LTL) extends the classical first-order logic to include temporal properties like "until", "as long as", and "at some point", etc. To satisfy an LTL formula, the model must produce a sequence of states that make it 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. A simple way to introduce real-time is to make the time discrete (or slotted), which allows for operators to move to the previous or next time slot. However, a single time slot is inadequate since the activities can occur over a wide range of time scales. We address this Pavana et al. [18] by defining time-slots of a few different durations. The extended SMT model can be solved using the NuXMV model checker proposed in Cavada et al. [93]. The result returned by NuXMV either indicates satisfiability (i.e., no conflict) or a set of counter-examples that can be used for conflict resolution.

6.2. Automated Conflict Resolution

For conflict resolution, in Pavana et al. [18], we considered the approach of perturbing the ORs minimally to eliminate the conflict. Often, the ORs involve some thresholds that can be tweaked without really changing their nature. For example, consider the following rules:

HVAC: if [(Temp>110F ∧ Firealarm_on>2 min) ∨ (CO2_level>15% ∧ Firealarm_on>2 min)], turnon_sprinkler

Security: if (water_level>15%), turnoff_sprinkler

and the safety property:

if (sprinkler_on >3 min), turnoff_sprinkler

Here, the conflict can be avoided by lengthening the duration of the sprinkler-on time. We attempt to resolve conflicts by "weakening" the operation rules. Naturally, there are situations in which tweaking the rules is undesirable or does not work, in which case, the resolution must rely on blocking the action with the lowest priority.

A proactive approach usually requires an evaluation of events that may occur in the next few time slots (each of duration τ) and the relevant "theories" as discussed above for feasible real-time predictions. It is possible to adjust thresholds and durations in the event of a conflict. Human input can make these modifications permanent. If an unexpected event occurs, it can be handled the same way as a standard dynamic resolution, i.e., by blocking one of the offending events. The lookahead period τ requires a consideration of the tradeoff between prediction reliability and flexibility in resolving the conflicts.

Conflict resolution via a minimal perturbation of the conditions involved in the ORs is a particularly difficult optimization problem that was not considered in the past. Viewed in terms of Boolean satisfiability, the perturbation does not simply flip the truth value of certain variables; instead, it needs to go down to the next level of detail and consider the perturbation of the underlying condition. For example, consider a clause such as (temp $>$ 25C $\vee$ time $<$ 5 min). In Boolean terms, this would be represented as $(x_1 \vee x_2)$ where x_1 = temp$>$ 25C, and x_2 = time $<$ 5 min. Here, we need to perturb the underlying values 25 and 5 to alter the truth value of the clause. In contrast, the methods for solving the highly popular weighted partial maxsat (WPM) problem in Ansotegui et al. [94] only minimizes the total weight of the falsified clauses.

6.3. Combinatorial-Optimization-Based Approach

We address this problem using a combinatorial optimization approach, which is popularly used in "incomplete" methods of solving WPM problems mentioned above. A survey of such methods is provided in the book [95] and the annual maxsat competition [96].

Numerous combinatorial optimization methods were explored in the literature conducted by Bianchi et al. [97]. The performance of these varies widely, with only a few algorithms being both efficient and successful in finding good solutions. In particular, Simulated Annealing and Dynamically Dimensioned Search (DDS) presented in Tolson et al. [98] appear to work the best in most cases [99]. We used DDS as it can handle high-dimensional problems very well. The basic DDS randomly chooses a subset of variables to perturb. We improved it by making use of "domain knowledge", i.e., the relative ranking of the variables in terms of their suitability for perturbation. We showed this to work extremely well in previous work, e.g., Pavana et al. [18] and Negar et al. [100].

The satisfiability of the modified formula is the constraint in the proactive approach, and hence the overall problem is defined as a constrained optimization problem. Several methods were explored and detailed in the survey paper Mezuramontesa et al. [101]. We selected the epsilon-constraint satisfaction method since it attempts to balance between feasibility and optimality when moving around in the state-space, and it appears to work well in practice. A cost is then added to determine how the parameters should be changed. The details may be found in Pavana et al. [18].

The evaluation in Pavana et al. [18] was performed using Home-IO simulator developed by Riera et al. [102], which provided a very rich set of sensors/actuators in a smart home and simplified modeling of real-time heat transfer (including radiation, convection, and conduction), air mass transfer, the effect of opening/closing doors/windows on temperature and luminance, the impact of wind outside the house, the impact of cloud cover, etc. We used Home-IO to emulate five different IoT subsystems, each with one controller and many sensors and actuators. These are (a) *Lighting Management*, (b) *Fire and Safety Management*, (c) *Security Management* (entry/exit of people), (d) *Surveillance and Monitoring* (unlawful activities of people inside/outside the home), and (e) *Climate Control*. There are many dependencies among these. The overall system has 50 ORs (of which 19 are perturbable) and 5 SPs.

One example of the sequence of events, including the conflicts and conflict resolution, is as follows: a fire in the kitchen results in smoke, which turns on the smoke detector, and kitchen temperatures go above 100F after a while, which activates the sprinkler. As a result, water flows and causes flooding (before the fire is extinguished), and consequently, water is shut off. This creates a "conflict" that can be resolved by changing the water-level threshold. Moreover, the kitchen AC is switched on. The high smoke level causes the LR and kitchen windows to be opened and the main door to be unlocked. This cools the LR, and its heater is turned on, which results in another conflict (AC and heater on simultaneously) but can be resolved by changing the thresholds. Meanwhile, an intruder enters through the unlocked door, which causes an intruder alarm. However, since the user is inside the home, surveillance cameras are disabled, but the intruder alarm requires recording the video. This conflict is resolved by priority-based resolution.

For these experiments, we use simplified physics so that the processing time is mostly dominated by satisfiability checking rather than by physics. We evaluate our algorithm's performance by considering three different perturbations of each perturbable rule (19 rules out of the original 50 ORs), yielding a total of 57 different possible perturbations.

Figure 3 illustrates the relationship between the number of rules and the perturbation-eligible fraction of these rules and the running time of the proposed proactive approach. Two additional cases with 100 and 200 rules are examined for this purpose (in addition to the original 50). By considering three different perturbations of each perturbable rule, we obtain a total of 57, 114, and 228 possible perturbations in the 50, 100, and 200 rule cases, respectively. In Figure 3, we consider varying fractions of these possible perturbations in Figure 3, i.e., for the case of 200 rules, the fraction 0.4 means that (0.4×228) perturbations are considered.

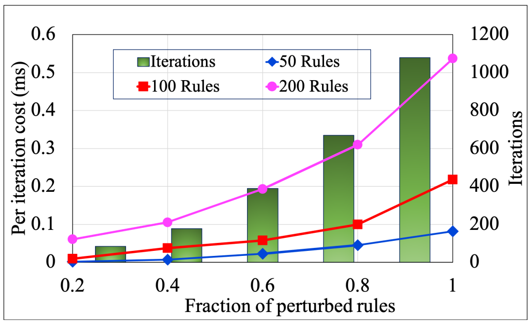

Figure 3. Number of iterations and per iteration cost vs. fraction of perturbed rules.

The left Y-axis represents the per-iteration cost, which increases as the fraction of perturbable rules and the total number of rules increases. The figure shows a nonlinear but relatively slow growth. The number of iterations, represented by bars with the right Y-axis scale, also grows nonlinearly but slightly faster. We believe the mechanism can easily scale to thousands of rules with a well-equipped server, despite our tests being conducted on a modest consumer desktop computer. A simplified physics assumption should suffice in most cases.

Additionally, perturbing the thresholds associated with the ORs does not always resolve the conflict; instead, it essentially delays its occurrence. As the system size increases, this is increasingly unlikely, and thus the perturbation becomes particularly helpful. However, note that any well-designed system automated system must include two provisions: (a) ability of user/administrator to override automated decisions concerning the actions and the state of the actuators, and (b) safety thresholds, which, if crossed, would automatically initiate corrective action regardless of the conflicts (and possibly disabling of the less critical actions). If the system lacks automated correction ability, it should be designed to provide an alert to the user to initiate overriding. Note that (b) is similar to the priority-based resolution discussed in Perumal et al. [28], Celik et al. [22] and Chaki et al. [68], except that the priorities are dynamic and context-dependent, rather than static.

7. Future Challenges in Conflict Detection and Resolution

Despite a large amount of work on the topic surveyed in this paper, numerous challenges remain. This section provides an overview of these.

A significant gap in previous works is represented by the inability to resolve conflicts proactively. The proposed proactive conflict detection and resolution strategy are discussed in detail in the Section 6. In contrast, the necessity of such an approach is discussed in Section 2.1. Celik et al. [61] and Nguyen et al. [63] considered a static analysis to check whether a collection of IoT apps working together adheres to the identified safety, security, and functional properties. Perumal et al. [28], Celik et al. [22], and Chaki et al. [68] adapt a dynamic conflict resolution strategy by analyzing the run time behavior of an IoT application and blocking the action of an IoT app that violate defined safety and security policies. The primary drawback of such an approach is that it may have undesirable physical consequences. For instance, suppose that a door should be unlocked only for a security service when the user is away for a vacation. A policy that disables the unlock-door state to resolve a conflict prevents the security service from entering the house, which may or may not be desirable depending on the circumstances. Much of the work that dynamically resolves conflicts (e.g., Nagendra et al. [79], Hsu et al. [19] and Celik et al. [22]) allows the users to specify policies that govern the collective behavior of the IoT applications. This presents a significant challenge in a complex IoT environment. An incorrect policy specification may prevent legitimate states, fail to block unsafe and insecure states or conflict with another policy. However, when a proactive conflict analysis strategy is used, the operational policies or rules defined for each subsystem are checked for the occurrence of intra-subsystem conflicts.

One major issue that remains unaddressed mainly is the consideration of cross-party accessibility in conflict detection and resolution. In our recent work [103], we considered a particular problem of providing minimal access in a fully trusted environment. We perform this in three phases: the first phase is called Authorization, where we explore approaches to efficiently assign enforcing parties to each SP in the system based on an authorization cost metric. In Phase 2, we build the access-control infrastructure, where each enforcer and the regular party must be provided the appropriate access lists, and Phase 3 uses these access lists to request and enforce access control at run-time. In this context, there is scope for much further work with regard to the tradeoffs between visibility vs. conflict detection/resolution and the corresponding efficient mechanisms to manage accesses.

Assuming that the basic interfaces are available so that any kind of accessibility is possible across subsystems, one could consider a variety of access control models, each with a different tradeoff between the detection/resolution of conflicts and the potential risks to the system. This also depends on the trust model across parties, ranging from complete trust to complete distrust. Note that the security risks persist even if the parties themselves are entirely trustworthy since, without any access control, a hack into one party could provide complete system access to the hacker. Thus, regardless of the trust model, it helps limit access to only necessary ones. Since the accesses are required only in specific circumstances, they should ideally be managed dynamically.

The other extreme situation is complete distrust between parties. Recently, Blockchain was touted as a mechanism for cooperation among nontrusting parties. For example, Ouaddah et al. [104] presents a decentralized access control framework for IoT based on Blockchain. Unfortunately, Blockchain requires broad disclosure of all data/actions to drive the consensus, and yet the only way to prevent a party from providing incorrect information is to have a redundant sensor in areas operated by other parties, which is impractical. Other methods will also face a similar problem—there is no way to verify sensor/actuator state without independent sensing by multiple parties.

In addition to these two extremes, several other models are possible and need to be explored. For example, all the "read" data (sensor values and actuator state) may be trusted, but additional verification may be needed for cross-party actuation requests. Detecting anomalies in such requests is an area for further work and may exploit learnings from the history or domain-specific sanity checks.

In addition to access control, there are several remaining challenges in conflict handling. One of them concerns the classification of conflicts in terms of their impact on the system's functioning. As discussed earlier, a "conflict" can be defined in many ways, and in general, it merely indicates something undesirable. The impact of the conflicts can range from minor resource inefficiency (e.g., lights being on when no-one is around), to user inconvenience (e.g., temperature or luminance moderately outside the comfortable range), to detrimental (e.g., unauthorized entry allowed), to disastrous (e.g., a person trapped during the fire). Learning such a classification automatically remains a challenge. A related issue is that some of the priorities may depend on the context rather than being fixed.

A fundamental question that still needs more work is defining or characterizing the conflicts. As discussed earlier, many characterizations exist, and they all have pros and cons. For example, defining conflicts via (violation of) safety properties is very general. We can consider almost any type of requirement as a safety property and then enforce it using the methods discussed in this paper. Yet, it raises two questions: (a) how do we come up with safety properties, and (b) how do we know whether we covered all essential safety properties? Neither of these questions are well-formed, and thus cannot be answered without further specification.

As discussed in Section 3 earlier, many authors characterized conflicts in terms of generic relationships between commands issued to one or more actuators. For example, Section 3.1.1 speaks of what is allowed across two controllers acting simultaneously or during a short period. While some of these are necessarily conflicts (and thus can always be considered safety properties), others may only represent potential conflicts, i.e., necessary but not sufficient conditions for conflict. For example, turning the light switch on and off repeatedly within a short period of time is only a potential conflict and may or may not be recognized as a real conflict. Nevertheless, a comprehensive specification of such generic situations can help identify or verify the safety properties. For example, if a specified SP does not result in any of these generic situations, then it may be regarded as illegitimate. Moreover, suppose the system does enter a state that, in retrospect, is considered to be undesirable. In that case, it is helpful to consider which of these generic properties are violated and accordingly formulate a new SP.

Yet another issue is that the SPs are often context-dependent; in fact, what is normally considered as a conflict (HVAC running and windows open) may be irrelevant during an emergency or other special situations (e.g., when there is a fire). Similarly, new SPs may become relevant during such special situations (e.g., ensure that elevators are locked). Since every SP carries a precondition for the specified postcondition, it is possible to specify the context in the precondition. This would result in very complex SPs; instead, it may be preferable to address infrequently occurring situations by adding, removing, or changing some of the SPs (usually a small number compared with the entire set of SPs). Addressing such changes remains a challenge.

8. Conclusions

Maintaining a safe and secure operation in large-scale Internet of Things systems is critical, as their functionality is dependent on the entrusted automation. Among the numerous types of problems that can occur in IoT systems, a significant subset is the conflicting operation of actuators, typically caused by interactions between automation services and system policies. This survey discusses various types of IoT system conflicts and their characterization. It then discusses several approaches to static and dynamic analysis of the conflicts. It also describes novel proactive methods for addressing conflicts. The survey also discusses many of the remaining challenges in conflict analysis in large IoT systems, particularly the multiparty systems where access-control and security issues become critical.

Funding: This research was funded by NSF grant CNS-2011252.

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: Data sharing not applicable.

Conflicts of Interest: The authors declare no conflicts of interest.

References

1. Strategy Analytics. IoT Strategies. Available online: www.strategyanalytics.com (acessed on 30 December 2014).
2. Evans, D. The Internet of Things—How the Next Evolution of the Internet Is Changing Everything. Available online: www.cisco.com (accessed on 30 December 2014).
3. Ibrahim, H.; Khattab, S.; Elsayed, K.; Badr, A.; Nabil, E. A formal methods-based Rule Verification Framework for end-user programming in campus Building Automation Systems. *Build. Environ.* **2020**, *181*, 106983. [CrossRef]
4. Ibrahim, H.; Hassan, H.; Nabil, E. A conflicts' classification for IoT-based services: A comparative survey. *PeerJ Comput. Sci.* **2021**, *7*, e480. [CrossRef] [PubMed]
5. Kho, J. The Next Wave in Lighting and the Internet of Things. Available online: www.forbes.com (accessed on 30 December 2014).
6. Philips. Hue Personal Wireless Lighting. Available online: www2.meethue.com (accessed on 30 December 2014).
7. Nest. The Brighter Way to Save Energy. Available online: https://nest.com/ (accessed on 30 December 2014).
8. Farber, D. Dacor Bakes Android Tablet into Wall Oven. Available online: http://www.cnet.com/ (accessed on 30 December 2014).
9. Schramm, M. CES Unveiled: The HAPIfork Aims to Help You Track Your Eating Habits with Bluetooth. Available online: http://www.tuaw.com/ (accessed on 30 December 2014).
10. Gizmag. Butterfleye Smart Surveillance Camera Keeps Watch with Your iPhone. Available online: http://www.gizmag.com (accessed on 30 December 2014).
11. Samsung. Smart Home. Intelligent Living. Available online: www.kickstarter.com/ (accessed on 30 December 2014).

12. Supermechanical. Listen to Your Home Wherever You Are. Available online: supermechanical.com/twine/ (accessed on 30 December 2014).
13. Vipersecurity. Control, Track, Locate, and Start Your Car from Virtually Anywhere with Your Smartphone. Available online: www.vipersecurity.com.au/ (accessed on 30 December 2014).
14. Haj-Assaad, S. QNX Previews Tesla-Sized Touch Screen in a Bentley. Available online: www.autoguide.com/ (accessed on 30 December 2014).
15. Vermesan, O.; Friess, P. Internet of Things—Converging Technologies for Smart Environments and Integrated Ecosystems. Available online: www.internet-of-things-research.eu (accessed on 30 December 2014).
16. Information Communication Development Authority of Singapore. The Internet of Things. Available online: www.ida.gov.sg (accessed on 30 December 2014).
17. Al-Farooq, A.; Al-Shaer, E.; Kant, K. A Formal Method for Detecting Rule Conflicts in Large Scale IoT Systems. In Proceedings of the IFIP/IEEE International Symposium on Integrated Network Management (IM 2019), Washington, DC, USA, 8–12 April 2019.
18. Pradeep, P.; Pal, A.; Kant, K. Automating Conflict Detection and Mitigation in Large-Scale IoT Systems. In Proceedings of the 2021 IEEE/ACM 21st International Symposium on Cluster, Cloud and Internet Computing (CCGrid), Melbourne, Australia, 10–13 May 2021 .
19. Hsu, K.H.; Chiang, Y.H.; Hsiao, H.C. Safechain: Securing trigger-action programming from attack chains. *IEEE Trans. Inf. Forensics Secur.* **2019**, *14*, 2607–2622. [CrossRef]
20. Ma, M.; Preum, S.M.; Stankovic, J.A. Cityguard: A watchdog for safety-aware conflict detection in smart cities. In Proceedings of the Second International Conference on Internet-of-Things Design and Implementation, Pittsburgh, PA, USA, 18–21 April 2017; pp. 259–270.
21. Liu, R.; Wang, Z.; Garcia, L.; Srivastava, M. RemedioT: Remedial actions for internet-of-things conflicts. In Proceedings of the 6th ACM International Conference on Systems for Energy-Efficient Buildings, Cities, and Transportation, New York, NY, USA, 13–14 November 2019 ; pp. 101–110.
22. Celik, Z.B.; Tan, G.; McDaniel, P.D. *IoTGuard: Dynamic Enforcement of Security and Safety Policy in Commodity IoT*; NDSS: San Diego, CA, USA, 2019.
23. Shah, T.; Venkatesan, S.; Ngo, T.; Neelamegam, K. Conflict detection in rule based IoT systems. In Proceedings of the 2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON), Vancouver, BC, Canada, 17–19 October 2019 ; IEEE: Piscataway, NJ, USA, 2019; pp. 0276–0284.
24. Chaki, D.; Bouguettaya, A.; Mistry, S. A Conflict Detection Framework for IoT Services in Multi-resident Smart Homes. *arXiv* **2020**, arXiv:cs.CY/2004.12702.
25. Abusafia, A.; Bouguettaya, A. Reliability Model for Incentive-Driven IoT Energy Services. *arXiv* **2021**, arXiv:cs.DC/2011.06159.
26. Lakhdari, A.; Bouguettaya, A.; Mistry, S.; Neiat, A.G.; Suleiman, B. Elastic Composition of Crowdsourced IoT Energy Services. *arXiv* **2020**, arXiv:cs.DC/2011.06771.
27. Huang, B.; Dong, H.; Bouguettaya, A. Conflict Detection in IoT-based Smart Homes. *arXiv* **2021**, arXiv:2107.13179.
28. Perumal, T.; Sulaiman, M.N.; Datta, S.K.; Ramachandran, T.; Leong, C.Y. Rule-based conflict resolution framework for Internet of Things device management in smart home environment. In Proceedings of the 2016 IEEE 5th Global Conference on Consumer Electronics, Kyoto, Japan, 11–14 October 2016 ; IEEE: Piscataway, NJ, USA, 2016; pp. 1–2.
29. Oh, H.; Ahn, S.; Choi, J.K.; Yang, J. Mashup service conflict detection and visualization method for Internet of Things. In Proceedings of the 2017 IEEE 6th global conference on consumer electronics (GCCE), Nara, Japan, 9–12 October 2018; IEEE: Piscataway, NJ, USA, 2017; pp. 1–2.
30. Sun, Y.; Wang, X.; Luo, H.; Li, X. Conflict detection scheme based on formal rule model for smart building systems. *IEEE Trans. Hum.-Mach. Syst.* **2014**, *45*, 215–227. [CrossRef]
31. Ahmed, M.O.; Elfaki, S.E.E. Adaptation Conflicts of Heterogeneous Devices in IOT Smart-Home. *Am. Acad. Sci. Res. J. Eng. Technol. Sci.* **2021**, *81*, 64–78.
32. Nakamura, M.; Igaki, H.; Matsumoto, K.I. Feature interactions in integrated services of networked home appliances. In Proceedings of the International Conference on Feature Interactions in Telecommunication Networks and Distributed Systems (ICFI'05) , Leicester, UK, 28–30 June 2005; pp. 236–251.
33. Leelaprute, P.; Matsuo, T.; Tsuchiya, T.; Kikuno, T. Detecting feature interactions in home appliance networks. In Proceedings of the 2008 Ninth ACIS International Conference on Software Engineering, Artificial Intelligence, Networking, and Parallel/Distributed Computing, Phuket, Thailand, 6–8 August 2008 ; IEEE: Phuket, Thailand, 2008; pp. 895–903.
34. Leelaprute, P. Resolution of feature interactions in integrated services of home network system. In Proceedings of the 2007 Asia-Pacific Conference on Communications, Bangkok, Thailand, 18–20 October 2007 ; IEEE: Bangkok, Thailand, 2007; pp. 363–366.
35. Igaki, H.; Nakamura, M. Modeling and detecting feature interactions among integrated services of home network systems. *IEICE Trans. Inf. Syst.* **2010**, *93*, 822–833. [CrossRef]
36. Li, X.; Zhang, L.; Shen, X. DIAC: An Inter-app Conflicts Detector for Open IoT Systems. *ACM Trans. Embed. Comput. Syst. (TECS)* **2020**, *19*, 1–25. [CrossRef]
37. Camacho, R.; Carreira, P.; Lynce, I.; Resendes, S. An ontology-based approach to conflict resolution in Home and Building Automation Systems. *Expert Syst. Appl.* **2014**, *41*, 6161–6173. [CrossRef]
38. IFTTT PLATFORM. Available online: https://platform.ifttt.com/docs#introduction (accessed on 15 February 2020).

39. Cabitza, F.; Fogli, D.; Lanzilotti, R.; Piccinno, A. Rule-based tools for the configuration of ambient intelligence systems: A comparative user study. *Multimed. Tools Appl.* **2017**, *76*, 5221–5241. [CrossRef]
40. TASKER For Android. Available online: https://tasker.joaoapps.com/index.html (accessed on 10 October 2021).
41. BIPIO GRaph API. Available online: https://github.com/bipio-server/bipio/wiki (accessed on 10 October 2021).
42. WIGWAG SMARTHOME. Available online: https://www.wigwagapp.com/ (accessed on 10 October 2021).
43. ZIPATILE2. Available online: https://www.zipato.com/ (accessed on 10 October 2021).
44. Sun, Q.; Yu, W.; Kochurov, N.; Hao, Q.; Hu, F. A multi-agent-based intelligent sensor and actuator network design for smart house and home automation. *J. Sens. Actuator Netw.* **2013**, *2*, 557–588. [CrossRef]
45. Luo, H.; Wang, R.; Li, X. A rule verification and resolution framework in smart building system. In Proceedings of the 2013 International Conference on Parallel and Distributed Systems, Seoul, Korea, 15–18 December 2013; IEEE: Seoul, Korea, 2013; pp. 438–439.
46. Maternaghan, C.; Turner, K.J. Policy conflicts in home automation. *Comput. Netw.* **2013**, *57*, 2429–2441. [CrossRef]
47. Newcomb, J.L.; Chandra, S.; Jeannin, J.B.; Schlesinger, C.; Sridharan, M. IOTA: A calculus for internet of things automation. In Proceedings of the 2017 ACM SIGPLAN International Symposium on New Ideas, New Paradigms, and Reflections on Programming and Software, Vancouver, BC, Canada, 25–27 October 2017 ; pp. 119–133.
48. Bak, N.; Chang, B.M.; Choi, K. Smart block: A visual programming environment for smartthings. In Proceedings of the 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC), Tokyo, Japan, 23–27 July 2018; Volume 2, pp. 32–37.
49. Sen, J. *Internet of Things: Technology, Applications and Standardization*; BoD–Books on Demand: Norderstedt, Germany, 2018.
50. Xie, W.; Zhu, H.; Wu, X.; Vinh, P.C. Formal verification of mCWQ using extended Hoare logic. *Mob. Netw. Appl.* **2019**, *24*, 134–144. [CrossRef]
51. Shehata, M.; Eberlein, A.; Fapojuwo, A. Using semi-formal methods for detecting interactions among smart homes policies. *Sci. Comput. Program.* **2007**, *67*, 125–161. [CrossRef]
52. Shehata, M.; Eberlein, A.; Fapojuwo, A.O. A taxonomy for identifying requirement interactions in software systems. *Comput. Netw.* **2007**, *51*, 398–425. [CrossRef]
53. Carreira, P.; Resendes, S.; Santos, A.C. Towards automatic conflict detection in home and building automation systems. *Pervasive Mob. Comput.* **2014**, *12*, 37–57. [CrossRef]
54. Android Things Website. Available online: https://developer.android.com/things (accessed on 10 October 2021).
55. SAMSUNG Smartthings. Available online: https://www.samsung.com (accessed on 10 October 2021).
56. Apple Homekit 2021. Available online: https://www.apple.com/shop/accessories/all/homekit (accessed on 10 October 2021).
57. OpenHAB 2021. Available online: https://www.openhab.org/ (accessed on 10 October 2021).
58. Shen, X.; Zhang, L.; Li, X. *A Systematic Examination of Inter-App Conflicts Detections in Open IoT Systems*; Technical Report; North Carolina State University, Department of Computer Science: Raleigh, NC, USA, 2017.
59. Liang, C.J.M.; Karlsson, B.F.; Lane, N.D.; Zhao, F.; Zhang, J.; Pan, Z.; Li, Z.; Yu, Y. SIFT: Building an internet of safe things. In Proceedings of the 14th International Conference on Information Processing in Sensor Networks, Seattle, WA, USA, 14–16 April 2015; pp. 298–309.
60. Trimananda, R.; Aqajari, S.A.H.; Chuang, J.; Demsky, B.; Xu, G.H.; Lu, S. Understanding and automatically detecting conflicting interactions between smart home IOT applications. In Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering, Online, 8–13 November 2020; pp. 1215–1227.
61. Celik, Z.B.; McDaniel, P.; Tan, G. Soteria: Automated IOT safety and security analysis. In Proceedings of the 2018 USENIX ATC, Boston, MA, USA, 11–13 July 2018; pp. 147–158.
62. Munir, S.; Stankovic, J.A. Depsys: Dependency aware integration of cyber-physical systems for smart homes. In Proceedings of the 2014 ACM/IEEE International Conference on Cyber-Physical Systems (ICCPS), Berlin, Germany, 14–17 April 2014; pp. 127–138.
63. Nguyen, D.T.; Song, C.; Qian, Z.; Krishnamurthy, S.V.; Colbert, E.J.; McDaniel, P. IotSan: Fortifying the safety of IoT systems. In Proceedings of the 14th International Conference on emerging Networking EXperiments and Technologies, Heraklion, Greece, 4–7 December 2018; pp. 191–203.
64. Ding, W.; Hu, H. On the safety of IOT device physical interaction control. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 832–846.
65. Ma, M.; Stankovic, J.A.; Feng, L. Cityresolver: A decision support system for conflict resolution in smart cities. In Proceedings of the 2018 ACM/IEEE 9th International Conference on Cyber-Physical Systems (ICCPS), Porto, Portugal, 11–13 April 2018; IEEE: Porto, Portugal, 2018; pp. 55–64.
66. Ma, M.; Stankovic, J.A.; Feng, L. Runtime monitoring of safety and performance requirements in smart cities. In Proceedings of the 1st ACM Workshop on the Internet of Safe Things, Delft, The Netherlands, 5 November 2017; pp. 44–50.
67. Ma, M.; Preum, S.M.; Stankovic, J.A. Demo abstract: Simulating conflict detection in heterogeneous services of a smart city. In Proceedings of the 2017 IEEE/ACM Second International Conference on Internet-of-Things Design and Implementation (IoTDI), Pittsburgh, PA, USA, 18–21 April 2017; pp. 275–276.
68. Chaki, D.; Bouguettaya, A. Adaptive priority-based conflict resolution of IoT services. In Proceedings of the 2021 IEEE International Conference on Web Services (ICWS), Chicago, IL, USA, 5–10 September 2021; pp. 663–668.
69. Berners-Lee, T.; Hendler, J.; Lassila, O. The semantic web. *Sci. Am.* **2001**, *284*, 34–43. [CrossRef]

70. Grau, B.C.; Horrocks, I.; Motik, B.; Parsia, B.; Patel-Schneider, P.; Sattler, U. OWL 2: The next step for OWL. *J. Web Semant.* **2008**, *6*, 309–322. [CrossRef]
71. Bouquet, P.; Giunchiglia, F.; Van Harmelen, F.; Serafini, L.; Stuckenschmidt, H. Contextualizing ontologies. *J. Web Semant.* **2004**, *1*, 325–343. [CrossRef]
72. Horrocks, I.; Patel-Schneider, P.F.; Boley, H.; Tabet, S.; Grosof, B.; Dean, M. SWRL: A semantic web rule language combining OWL and RuleML. *W3C Memb. Submiss.* **2004**, *21*, 1–31.
73. Celik, Z.B.; Babun, L.; Sikder, A.K.; Aksu, H.; Tan, G.; McDaniel, P.; Uluagac, A.S. Sensitive information tracking in commodity IoT. In Proceedings of the 27th USENIX Security Symposium (USENIX Security 18), Baltimore, MD, USA, 15–17 August 2018; pp. 1687–1704.
74. Ma, M.; Preum, S.M.; Tarneberg, W.; Ahmed, M.; Ruiters, M.; Stankovic, J. Detection of runtime conflicts among services in smart cities. In Proceedings of the 2016 IEEE International Conference on Smart Computing (SMARTCOMP), St. Louis, MO, USA, 18–20 May 2016 ; IEEE: Piscataway, NJ, USA, 2016; pp. 1–10.
75. Coppers, S.; Vanacken, D.; Luyten, K. FORTNIoT: Intelligible Predictions to Improve User Understanding of Smart Home Behavior. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* **2020**, *4*, 1–24. [CrossRef]
76. Liang, T.; Zeng, B.; Liu, J.; Ye, L.; Zou, C. An unsupervised user behavior prediction algorithm based on machine learning and neural network for smart home. *IEEE Access* **2018**, *6*, 49237–49247. [CrossRef]
77. Du, Y.; Lim, Y.; Tan, Y. A novel human activity recognition and prediction in smart home based on interaction. *Sensors* **2019**, *19*, 4474. [CrossRef]
78. Wu, S.; Rendall, J.B.; Smith, M.J.; Zhu, S.; Xu, J.; Wang, H.; Yang, Q.; Qin, P. Survey on prediction algorithms in smart homes. *IEEE Internet Things J.* **2017**, *4*, 636–644. [CrossRef]
79. Nagendra, V.; Bhattacharya, A.; Yegneswaran, V.; Rahmati, A.; Das, S.R. VISCR: intuitive & conflict-free automation for securing the dynamic consumer IOT infrastructures. *arXiv* **2019**, arXiv:1907.13288.
80. Chi, H.; Zeng, Q.; Du, X.; Yu, J. Cross-app interference threats in smart homes: Categorization, detection and handling. In Proceedings of the 2020 50th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN), Valencia, Spain, 29 June–2 July 2020 ; IEEE: Piscataway, NJ, USA, 2020; pp. 411–423.
81. Fernandes, E.; Paupore, J.; Rahmati, A.; Simionato, D.; Conti, M.; Prakash, A. Flowfence: Practical data protection for emerging iot application frameworks. In Proceedings of the 25th USENIX security symposium (USENIX Security 16), Austin, TX, USA, 10–12 August 2016; pp. 531–548.
82. Bastys, I.; Balliu, M.; Sabelfeld, A. If this then what? Controlling flows in IoT apps. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 1102–1119.
83. Mohsin, M.; Anwar, Z.; Husari, G.; Al-Shaer, E.; Rahman, M.A. IoTSAT: A formal framework for security analysis of the Internet of Things. In Proceedings of the IEEE Conference on Communications and Network Security (CNS), Philadelphia, PA, USA, 17–19 October 2016; IEEE: Piscataway, NJ, USA, 2016; pp. 1–7.
84. Tian, Y.; Zhang, N.; Lin, Y.H.; Wang, X.; Ur, B.; Guo, X.; Tague, P. Smartauth: User-centered authorization for the internet of things. In Proceedings of the 26th USENIX Security Symposium (USENIX Security 17), Vancouver, BC, Canada, 16–18 August 2017; pp. 361–378.
85. Zhang, W.; Meng, Y.; Liu, Y.; Zhang, X.; Zhang, Y.; Zhu, H. Homonit: Monitoring smart home apps from encrypted traffic. In Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security, Toronto, ON, Canada, 15–19 October 2018; pp. 1074–1088.
86. Pandita, R.; Xiao, X.; Yang, W.; Enck, W.; Xie, T. WHYPER: Towards automating risk assessment of mobile applications. In Proceedings of the 22nd USENIX Security Symposium (USENIX Security 13), Washington, DC, USA, 14–16 August 2013; pp. 527–542.
87. Wang, J.; Hao, S.; Wen, R.; Zhang, B.; Zhang, L.; Hu, H.; Lu, R. IoT-praetor: Undesired behaviors detection for IoT devices. *IEEE Internet Things J.* **2020**, *8*, 927–940. [CrossRef]
88. Jia, Y.J.; Chen, Q.A.; Wang, S.; Rahmati, A.; Fernandes, E.; Mao, Z.M.; Prakash, A.; Unviersity, S. ContexloT: Towards Providing Contextual Integrity to Appified IoT Platforms. In Proceedings of the NDSS, San Diego, CA, USA, 26 February–1 March 2017; p. 2.
89. Wang, Q.; Hassan, W.U.; Bates, A.; Gunter, C. Fear and logging in the internet of things. In Proceedings of the Network and Distributed Systems Symposium, San Diego, CA, USA, 18–21 February 2018.
90. Babun, L.; Celik, Z.B.; McDaniel, P.; Uluagac, A.S. Real-time analysis of privacy-(un) aware IoT applications. *arXiv* **2019**, arXiv:1911.10461.
91. Ernst, M.D. *Static and Dynamic Analysis: Synergy and Duality*; MIT Computer Science & Artificial Intelligence Lab: Cambridge, MA, USA, 2003.
92. de Moura, L.; Bjørner, N. Z3: An Efficient SMT Solver. In *TACAS*; Ramakrishnan, C.R., Rehof, J., Eds.; Springer: Berlin/Heidelberg, Germany, 2008; pp. 337–340.
93. Cavada, R.; Cimatti, A.; Dorigatti, M.; Griggio, A.; Mariotti, A.; Micheli, A.; Mover, S.; Roveri, M.; Tonetta, S. The nuXmv symbolic model checker. In Proceedings of the 26th International Conference on Computer Aided Verification, Vienna, Austria, 18–22 July 2014; pp. 334–342.

94. Ansótegui, C.; Bonet, M.L.; Levy, J. A new algorithm for weighted partial MaxSAT. In Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, Atlanta, GA, USA, 1 December 2009–18 January 2010 .
95. Hoos, H.; Sttzle, T. *Stochastic Local Search: Foundations and Applications*; Morgan Kaufmann Publishers Inc.: San Francisco, CA, USA, 2004.
96. MAXSAT 2020. Available online: https://maxsat-evaluations.github.io/2020/ (accessed on 10 July 2020).
97. Bianchi, L.; Dorigo, M.; Gambardella, L.M.; Gutjahr, W.J. A survey on metaheuristics for stochastic combinatorial optimization. *Nat. Comput.* **2009**, *8*, 239–287. [CrossRef]
98. Tolson, B.; Shoemaker, C. Dynamically dimensioned search algorithm for computationally efficient watershed model calibration. *Water Resour. Res.* **2007**, *43*. [CrossRef]
99. Arsenault, R.; Poulin, A.; Côté, P.; Brissette, F. Comparison of stochastic optimization algorithms in hydrological model calibration. *J. Hydrol. Eng.* **2014**, *19*, 1374–1384. [CrossRef]
100. Mohammadi, N.; Sondur, S.; Kant, K. Effective Configuration Optimization of Large Scale Software Systems. 2022 . https://www.kkant.net/papers/ICFEC_2022_Effective_Configuration_Optimization_of_Data_Center_Services.pdf (accessed on 10 July 2020).
101. Mezura-Montes, E.; Coello, C. Constraint-handling in nature-inspired numerical optimization: Past, present and future. *Swarm Evol. Comput.* **2011**, *1*, 173–194. [CrossRef]
102. Riera, B.; Emprin, F.; Annebicque, D.; Colas, M.; Vigario, B. HOME I/O: A virtual house for control and STEM education from middle schools to Universities. *IFAC-Papers OnLine* **2016**, *49*, 168–173. [CrossRef]
103. Pradeep, P.; Kant, K.; Pal, A. Managing Access Control in Large-Scale Multi-Party IoT Systems. In Proceedings of the IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing (CCGrid 2022), Taormina, Sicily, Italy, 16–19 May 2022.
104. Ouaddah, A.; Abou Elkalam, A.; Ouahman, A.A. Towards a novel privacy-preserving access control model based on blockchain technology in IoT. In *Europe and MENA Cooperation Advances in Information and Communication Technologies*; Springer: Cham, Switzerland, 2017; pp. 523–533.