Available online at www.qu.edu.iq/journalcm
JOURNAL OF AL-QADISIYAH FOR COMPUTER SCIENCE AND MATHEMATICS
ISSN:2521-3504(online) ISSN:2074-0204(print)

Formal Verification Approach for IoT Conflict Resolution by Priority

Zinah Hussein Toman

Department of Computer Science, Faculty of Computer Science and Information Technology, Al-Qadisiyah University, Iraq. Email: zinah.hussein@qu.edu.iq

ARTICLE INFO

Article history:
Received: 22 /01/2026
Rrevised form: 10 /02/2026
Accepted : 12 /02/2026
Available online: 30 /03/2026

Keywords: Conflict detect, Conflict resolution, Event-B, IoT, Formal method.

ABSTRACT

Internet of Things (IoT)-based solutions, especially automation systems, are essential for controlling networked devices and creating adaptive and intelligent environments. End-user-developed IoT services/apps interact and share simultaneous access to devices depending on their preferences, thereby increasing safety, security, and correctness issues. System failures or reduced performance might result from the traditional reactive approach to dispute resolution, which identifies and fixes problems after they have already happened. To tackle important issues, IoT-based applications require special tools to check and analyse their design, helping to find problems and conflicts in the instructions and interactions of IoT applications. This paper employs the Event-B formal method to introduce a novel approach for detecting and resolution conflicts in IoT systems by using priority. To systematically identify and resolve any conflicts, formal models of the IoT system are developed and refined using abstraction and refinement techniques. The objective of this paper is to promote the use of formal Event-B modelling and verification during the initial stages of the development of an IoT system to expedite the detection and fixing of problems. The proposed model addresses several types of conflicts, such as resource, functional, priority, and policy conflicts. The Rodin platform automates verification, ensuring that all declared invariants and attributes are preserved during system operation. As an example of our method in action, we will take a look at an ECG IoT system. The model accurately detects possible conflicts in these situations. We verify our method using the Rodin model checking tool, proof obligations and the ProB animator.

MSC..

https://doi.org/10.29304/jqcsm.2026.18.12640

1. Introduction

The Internet of Things is now a widely used concept in many different fields. We should combine several IoT devices (IoT services) to offer the user full services to satisfy their complex needs. Conflicts are likely to arise because these services typically coexist. Furthermore, every user may have unique needs. Assuming multiple people are present in a smart home, a dispute could arise when they all ask for the same service. In actuality, a conflict may still arise even though they seek separate IoT services, provided that there is a functional impact [1,2]. Because these IoT systems depend on the trusted automation to function, maintaining their safe and secure operation is crucial. Conflicting actuator operation is a significant class of malfunctions that can arise in Internet of Things systems [3]. Unique circumstances, either impossible to predict or manage statically, typically generate these malfunctions. This phenomenon is especially true in situations where multiple independently designed and operated subsystems are working in parallel [4]. There are two fundamental approaches to resolving conflicts: (a) Avoidance, in which possible conflicts are found through static analysis (i.e., during design or initial configuration) and fixed by altering

∗Corresponding author Zinah Hussein Toman

Email addresses: zinah.hussein@qu.edu.iq

the system's operational policy logic; and (b) Resolution, in which conflicts are identified during runtime and remedial measures are implemented. However, in cyber-physical systems, detecting conflicts when they arise and resolving them reactively might be undesirable and even harmful because we essentially wait for an issue to arise rather than foresee and resolve it (if at all possible) [5].

Verification is an essential part of IoT systems. Verification aims to confirm that a system or component satisfies its requirements. This process ensures that all IoT systems and devices operate without errors. Methods for system verification include testing, simulation, and formal verification. Formal verification makes use of logical and mathematical methods to make sure the system satisfies its requirements [6]. For the development and implementation of successful strategies to ensure the safety of various applications, including smart homes, healthcare, infrastructure, and transportation systems, formal verification is necessary [7].

Event-B is a formal approach to incremental development based on set theory. Unlike other formal approaches such as Petri Nets and process algebra, Event-B distinguishes itself by including a refinement mechanism for managing system complexity. This is the main reason for using Event-B as a methodology in our work. The designers begin by abstractly describing the system's needs, focusing on its core goal and framework. They then steadily add functionality through enhancements [9]. Refinement is a fundamental concept in Event-B. This study introduces a novel approach to detecting and solving conflicts in IoT systems using a formal modelling language [8]. Proof obligations (POs) are the foundation of the Event-B approach, as they are logical requirements that must be proven true to verify the system [9-12].

This paper's primary goal is to use the Event-B formal approach to present a novel strategy for priority-based conflict detection and resolution in IoT systems. Below is a summary of this paper's objectives.

1. We propose a formal approach to modelling resource, functional, and priority conflicts for detecting and resolving them in IoT systems.

2.We will use the refinements in the detection and resolution model to enhance the accuracy of conflict detection and classification.

3. By using the priority-based resolution conflict to resolve and prevent the conflict, this ensures that the system becomes safer and error-free.

4. Formal verification of critical safety properties, such as priority compliance, capacity preservation, and conflict-free activation.

We used an electrocardiogram (ECG) IoT system in our model to verify the accuracy and performance of the IoT conflict resolution model. The structure of the paper is as follows: Section 2 introduces additional works pertinent to our methodology. In addition to providing guidelines for modelling, verification, and model requirements, Section 3 introduces the idea of IoT conflict resolution. A summary of the formal approach to Event-B is given in Section 4, IoT conflict resolution in the Event-B model is formalised in Section 5, our technique is suggested to be verified and validated in Section 6, and this work is concluded in Section 7.

2. Related Works

Researchers have been trying to comprehend IoT technology for many years. Nonetheless, the primary focus of these contributions is on conflict detection and resolution. Formal techniques, based on mathematical accuracy, are an effective tool for systematically identifying and resolving such conflicts prior to implementation. This subsection provides a literature review for several of these publications, organised historically.

A. Al Farooq et al. [13] provide a formal technique for an IoT Conflict Checker that is designed to assure the safety of controller and actuator behaviour in the event of a conflict, as well as the identification of any policy breaches. This technique specifies the safety protocols for actions, controllers, and trigger events. The proposed approach uses Prologue to demonstrate logical soundness and completion. Shehata et al. [14, 15] propose IRIS, a semi-formal approach, to identify policy conflicts and interactions in intelligent homes. To analyse policy elements that go beyond telecommunications, the model includes an interaction taxonomy. It also has a run-time module devoted to conflict detection and resolution, which uses simulation techniques to handle override conflicts as well as negative impact conflicts. Nevertheless, IRIS has certain limitations; notably, it does not take into account rule dependency, which restricts its ability to identify both direct and indirect dependence conflicts.

Hsu et al. [16] modelled interacting among devices and rules as a finite state machine (FSM), where device statuses represent states and automation rules denote transitions. The FSM is assessed against a collection of established security policies, and any breach of these policies is recognised as a vulnerability under the automation rules. Ma et al. in [17] further delineate safety requirements of a city and identify conflicts across smart city services arising from violations of these safety requirements.

Liu et al. [18] and Celik et al. [19] define conflicts as violations of policy. User-defined rules, like "if a user is away, then activate user-away mode," and application-specific policies, like "if the temperature is low, then activate the fan," are the two main categories into which conflicts fall. The disagreements are then divided into two categories: cyclic events and racing. The simultaneous occurrence of two or more events involving opposing activity is what defines racing events. Two or more events that are controlled by a certain set of circumstances and behaviours that constantly activate one another are referred to as cyclic events.
In contrast to previous studies, there has been insufficient research validating IoT conflict resolution systems using EventB; however, our methodology presents a novel formal verification technique based on the EventB model that defines IoT conflict resolution through prioritization.

3. IoT Conflict resolution requirements:

The system requirements outline the anticipated functionalities, which include adhering to service standards and maintaining operational limits. A requirement is a comprehensive statement that outlines the tasks a system must perform or the constraints on its execution [27]. This section outlines the structure the IoT conflict resolution based on priority. It shows how the system detects and resolution IoT conflicts.

3.1 The Structural requirements

Conflict in IoT happens when there are differences in rules, services, or data from connected devices, leading to problems like devices not working together (for example, turning on and off at the same time), inconsistent data, or subtle environmental issues (like the air conditioning running while a window is open). The main reasons for these problems are complex connections, systems that don't work well together, and different goals for each system that require advanced methods to find and fix issues. [28].

SR1- The system must define IoT devices and the conditions under which they operate.
SR2- Each device in the system may use or offer resources.
SR3- Devices should be pending or valid for alerting.
SR4- The system will categorise conflict kinds in a methodical manner.
SR5- No two conflicting devices may utilise the same unique resource.

3.2- Conflict detection requirements:

Operating several subsystems in the same physical location is a major cause of dependency and, consequently, conflicts. Since the subsystems may operate in opposing ways depending on the characteristics of the shared area that the sensors monitor (such as temperature, motion, brightness, etc.), there is a natural coupling between them. Keep in mind that the measured attribute is the dependent factor; it does not matter whether the measurement is conducted by a single sensor shared among all the subsystems or by independent sensors [4,5]. To satisfy all requirements, we developed our proposed approach.

Conf_Detc: Each conflict must be categorised under a specific conflict type.

This conflict occurs when utilising a classification scheme to categorise conflicts into resource, functional, priority, and policy categories. This classification framework facilitates systematic conflict resolution by categorising the nature of the conflict.
The conflict type is:
- **Resource Conflict:** Two services attempt to access the same exclusive device (e.g., a camera).
- **Functional conflict:** Occurs when two devices issue conflicting commands to one another, for example, when one device opens a window while another device attempts to close it.
- Priority Conflict: Both services are valid; however, one takes precedence over the other (e.g., emergency service versus comfort service).
- Policy Conflict: The request from the device violates established system guidelines or policies, such as excessive energy consumption.

3.3- Conflict Resolution by Priority:

Finding conflicts as they happen and fixing them afterward can be unwise and sometimes dangerous in cyber-physical systems, because this method means waiting for a problem to occur instead of trying to prevent it beforehand when possible. Conflicts can be identified from a wide range of angles, and their effects can vary from small discrepancies to significant accidents. Therefore, prompt conflict identification is crucial for resolving conflicts, which range from merely ignoring the issue to completely prohibiting specific behaviours or, conversely, initiating a counteraction. [29]. In cyber-physical systems, reactive resolution and conflict detection can be undesirable and even dangerous, as they wait for problems to occur instead of anticipating and resolving them when possible.

Prio1: The system should proactively assign resources to important devices.
Prio2: The system must prioritise important services or devices.

Prio3: Only when a device has a higher priority can it preempt another.

4 . The Event-B Method:

The B-method and the idea of action systems served as the foundation for the formal technique of discrete system modelling known as Event-B. Safety-critical systems make use of it [7]. Reactive systems must be constructed correctly by design in order for integer arithmetic, typed set theory, and first-order logic to function as intended. The ability to create system designs that are right by construction is a key feature of this approach. Once we have created the proper plan, the remaining task is to write or implement the appropriate code [6]. As seen in Figure 1, the Event-B paradigm consists of both static and dynamic elements, which can be conceptualised as CONTEXT and MACHINE.

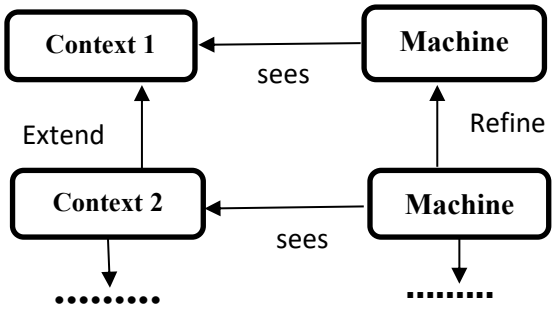

Figure1. An Event-B Architecture.

The CONTEXT FILE lists the fundamental characteristics of the model. This first-order theory supports sets (user-defined types) and numerical CONSTANT declarations. After expressing constants in a specific context, axioms and theorems establish their characteristics and relationships. An axiom is defined as an AXIOM that may be inferred from one axiom but not from another. THEOREMS outline characteristics that should be inferred from the guiding principles. The SEES clause imports constants and axioms from context into the MACHINE to explain the dynamic structure or system evolution. The machine needs a few EVENTS to specify the machine's anticipated state changes, a set of VARIABLES to describe the state of the system, and a set of INVARIANTS to guarantee consistency. [8-12]. The format used to express each event is as follows:

ANY parameters **WHEN** guards **THEN** actions **END**.

The guard specifies the prerequisite that must be met for an event to occur. The event is enabled in this state only if a parameter value keeps its guard in this state. Actions explain how state factors change in reaction to a trigger.

5.Formalization IoT Conflict Resolution Model in Event-B

In this section, we outline our approach to using the Event-B methodology to formalise the IoT resolution of conflicts as a priority.This enables rapid detection of errors and guarantees that the development process adheres to the specified criteria. We employed a refinement-based strategy that simulated IoT conflict resolution. It is feasible that by using this strategy, our model would be able to change from general ideas to precise specifications. As a result, the suggested model is made up of three sub-models: the abstract model, the first refined model (Conflict Detection), and the second refined model (Conflict Resolution). The refining principle, as illustrated in Figure 2, is responsible for enabling all of this.

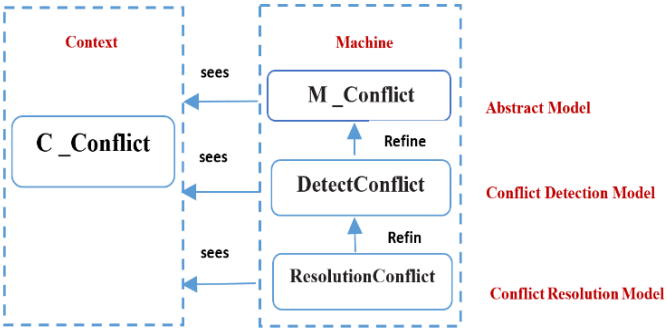

Figure 2. The proposed model of Event-B formal model for IoT Conflict Resolution.

5.1 Abstract model

This section presents the comprehensive procedure for building the abstract model. The abstract model represents the IoT conflict resolution framework, which established the essential system needs. The conceptual framework of the conflict detection and resolution paradigm consists of two distinct parts: the context (C_Conflict) and the machine (M_Conflict).

- **Context (C_Conflict):**
  C_Conflict serves as the context outlining the static architecture of the proposed IoT conflict-resolution mechanism. It delineates abstract sets for:
  - *DEVICE:* Collection of all IoT devices that can be managed by the system.
  - *RESOURCE*: Collection of resource types, including bandwidth, CPU, and energy.
  - *CONFLICT:* The classification of conflict types includes resource, functional, and priority categories.
  -*PRIORITY:* A collection of priority levels that are used for preemption and arbitration.

The constants that capture reso_req, reso_capacity, dev_critical, priority, and conflict_type are also included. To model and refine behaviour in Event B, the axioms that follow guarantee structural consistency and provide the necessary foundational assumptions. The logical assertions in the AXIOMS clause explain the attributes and connections of the entities listed in the SETS and CONSTANTS clause. (See Figure.3).

CONTEXT C_Conflict
SETS: DEVICE, RESOURCE, CONFLICT, PRIORITY
CONSTANTS: reso_req, reso_capacity, dev_critical, priority, conflict_type, prior_dev, Fun_dev
AXIOMS
axm1: dev_critical ⊆ DEVICE
axm2: reso_req ∈ DEVICE × RESOURCE → ℕ
axm3: reso_capacity ∈ RESOURCE → ℕ
axm4: priority ∈ DEVICE → PRIORITY
axm5: conflict_type ∈ DEVICE → CTYPE
axm6: PRIORITY ≠ ∅
axm7: prior_dev ⊆ PRIORITY
axm8: Fun_dev ⊆ DEVICE
END

Figure 3. An Event-B C_Conflict Context.

- **Machine (M _Conflict):**

M _Conflict is an abstract behavioural machine that simulates the runtime state of an IoT conflict resolution. It bears responsibility for most of the concepts defined in the conflict model. (that satisfy *SR1, SR2*, and *SR3*). Consequently, as illustrated in Figure 4, the machine (clause SEES) is viewed in the previously described context of *C_Conflict*, the first machine in the Event-B model for IoT conflict resolution.

MACHINE
M_Conflict
REFINES
Conflict_ Detect
SEES
C_Conflict
VARIABLES
……..
INVARIANTS
inv1: Active_dev ⊆ DEVICE
inv2: alert ⊆ DEVICE ∧ Pend ⊆ DEVICE
inv3: pend ⊆ DEVICE
inv4: alloc_Res ∈ DEVICE × RESOURSE → ℕ
inv5: conf ⊆ DEVICE ↔ DEVICE
inv6: Role ⊆ DEVICE
inv7: ∀r·r ∈ RESOURSE ⇒ Σ d·d ∈ DEV | alloc_Res(d,r) ≤ reso_capacity (r)
inv8: ∀d, r·d ∈ Active_dev ∧ d ∈ dev_critical ∧ r ∈ RESOURSE ⇒ alloc_Res (d, r) ≥ reso_req (d, r)
Inv9: ∀d1, d2· (d1 ∈ Active_dev ∧ d2 ∈ Active_dev ∧ (d1↦d2) ∈ Fun_dev ⇒(d1↦d2) ∈ conf
Inv10: ∀d1, d2·(d1↦d2) ∈ Fun_dev ⇒ ¬ (d1 ∈ Active_dev ∧ d2 ∈ Active_dev)
Inv11: ∀d1, d2·(d1↦d2) ∈ Role ⇒ (priority (d1), priority (d2)) ∈ prior_dev
inv12: alert ∈ dom(conf) ∪ ran(conf)

Figure 4. An Event-B M_Conflict Machine.

Subsequently, we established several variables to represent the elements of the abstract model. Nonetheless, the *inv1* is a type invariant that models the *SR1,* which signifies that only legitimate devices may be active, thereby preventing undefined or non-existent devices from participating. In this manner, *inv2* is used to model the *SR2*, ensuring that alerts and scheduling mechanisms are applicable to actual devices and *inv3* for devices waiting for service or activation, while inv4 models the *SR3*, defining the structural framework for resource sharing. The *inv5* and inv6 enforce a coherent conflict representation and delineate the role of preemption exclusively among valid devices, respectively. Furthermore, the *inv7* makes sure that no resource's allocation exceeds its limit. Additionally, *inv8* ensures that the devices receive the bare minimum of resources required when they are in use. Furthermore, *inv9* finds and logs unsafe devices within the conflict set. By stating that devices that are operationally incompatible cannot be used simultaneously, the *inv10* describes safety functions of conflicts. This invariant prevents dangerous system behaviour by making sure that opposing orders don't cause incompatible devices to operate simultaneously.

These invariants combine to provide robust safety guarantees for resource use and interference with their operations. These invariants model the requirement for *Conf_Detc2*.

By mandating that the preemption rule adhere to the context's prioritisation order, *inv11* formalises conflict resolution through priority awareness. Therefore, during the resolution of conflicts, only devices with a higher priority can preempt those with a lower priority. To make sure that only devices involved in unresolved conflicts receive notifications, the *inv12* further connects alerts to conflicts that are found. This guarantees consistent system notifications and conflict detection. Three events are included as abstract elements in this model to preserve its essential features. The following explains the three events: *ConflictDetect, AllocatePriori*ty, and *PreemptPriority.*

- The ***ConflictDetect*** event is in charge of identifying undesirable or dangerous system conditions that might lead to conflicts in the Internet of Things system before they occur. This event is activated exclusively when the grd1 verifies that *dev1* and *dev2* are distinct devices within the *DEVICE* set. The *grd2* subsequently determines whether a conflicting state exists. This conflict may be a functional conflict, where the devices are active despite being in opposition; a resource conflict, where the overall allocation of a resource is in excess of its capacity; or a priority conflict, where a higher-priority device is awaiting while a lower-priority device is functioning. The event is initiated when one of the above conditions is met. The actions add the pair (d1↦d2) and (d2↦d1) to the conflicts relation (*act1*) when the event is triggered to record the conflict symmetrically. They also incorporate both devices into the alerts set (*act2*) to indicate that resolution is necessary. As a result, all safety invariants are preserved, as the event cannot change the operational condition of devices or resources. This event preserves the *Conf_Detc* requirement (see Fgiure.5).

ConflictDetect ≙ **WHERE**

grd1: dev1 ∈ DEVICE ∧ dev2 ∈ DEVICE ∧ dev1 ≠ dev2
grd2: (dev1 ∈ Active_dev ∧ dev2 ∈ Active_dev ∧
(dev1↦dev2) ∈ Fun_dev) ∨ (∃r·r ∈ RESOURSE ∧ Σ e·e
∈ DEVICE | alloc_Res (e,r) > reso_capacity (r)) ∨
((priority (d2), priority (d1)) ∈ prior_dev ∧ dev2 ∈
Active_dev ∧ dev1 ∈ pend)
THEN
act1: conf ≔ conf ∪ {(dev1↦dev2),(dev2↦dev1)}
act2: alert ≔ alert ∪ {dev1, dev}
END

Figure. 5. The Event ConflictDetect.

- The ***PriorityAllocate*** event is responsible for the proactive allocation of resources to critical devices and the safe activation of these devices. By guaranteeing that critical devices receive at least the minimum amount of resources while adhering to resource capacity limitations, this approach avoids conflicts. This event satisfies the *Prio1* requirement (see Figure 6).

PriorityAllocate ≙ **WHERE**
grd1: dev ∈ DEVICE ∧ res ∈ RESOURSE
grd2: dev ∈ dev_critical
grd3: y ∈ ℕ ∧ y ≥ reso_req (dev, res)
grd4: Σ a·a ∈ DEVICE | (alloc_Res ⩤ {(dev↦res)
↦ y}) (a, res) ≤ reso_capacity (res)
THEN
act1: alloc_Res ≔alloc_Res ⩤ {(dev↦res) ↦ y}
act2: Active_dev ≔Active_dev ∪ {dev}
act3: pend ≔ pend \ {d}
END

Figure. 6. The Event PriorityAllocate

When *grd1* confirms that resource res and device dev are legitimate system components and *grd2* confirms that the device is classified as critical, the event is triggered. The variable y is guaranteed to be a natural number and to use the least number of resources possible by the *grd3*. The *grd4* makes sure that resource r's capacity is not exceeded while allocating y to the device. In these circumstances, the first action, *act1*, allocates y resources to the device, changing the allocation function. The device is added to the active set by the second action, *act2*, but it is taken out of the pending set by the third action, *act3*.

- The ***PreemptByPriority*** event resolves conflicts using priority-based control. The event enables a higher-priority device to take over from an active lower-priority device by safely allocating resources and changing device activity.This helps maintain system security and adherence to priority rules. This event satisfies the *Prio2* and *Prio2* requirements (see Figure.7).

The event is activated upon confirmation by *grd1* that device devH has the ability to preempt device devL in accordance with the established policy. The *grd2* and *grd3* maintain that the higher-priority device *devH* remains pending, while the lower-priority device *devL* is active. The grd4 identifies the affected resource. The *act1* then releases the resource from *devL* and reallocates it to *devH*, ensuring that devH receives the minimum necessary allocation. Additionally, *act2* deactivates devL and activates *devH* to update the active set. And *act3* eliminates the resolved conflict from their conflict relationship.

```
PreemptByPriority ≙ WHERE
    grd1: (devH↦devL) ∈ Role
    grd2: devH ∈ pend
    grd3: devL ∈ Active_dev
    grd4: res ∈ RESORSE
THEN
    act1: alloc_Res ≔ alloc_Res ⩤ {(devL↦res) ↦ 0} ⩤
          {(devH↦res) ↦ max (reso_req (devH, res),
          reso_req (devH, res))}
    act2: Active_dev ≔ (Active_dev \ {devL}) ∪{devH}
    act3: conf≔ conf \ ({(devH↦devL), (devL↦devH)} ∩ conf)
END
```

Figure. 7. The Event PreemptByPriority.

5.2 First Refinement (Conflict Detection Model)

This approach introduces the idea of conflict detection through a definition of safety attributes or a policy that outlines requirements to prevent illicit activity. We improved the previous model for conflict identification by incorporating more events. We propose a methodical series of actions to identify conflicts in IoT systems. The procedures are made to detect situations in which competing tasks are suggested for the same resource or device within a given amount of time. The phases encompass essential components that are crucial for relating to each occurrence of our model. This model also preserves the *Conf_Detc* requirement.

- **Machine (DetectConflict):**

The *DetectConflict* machine refines *M_Conflict* by increasing its degree of conflict detection while preserving all abstract behaviours. It adds the variable Buffer in *inv1* for recording recent device–resource abnormalities and the variable *ConfType* in *inv2* to categorise each detected conflict. The associated invariants guarantee that abnormality records and conflict classes are consistently well-defined. This refinement does not change device states or resource allocations; rather, it observes and indicates conflicts. Consequently, the abstract model maintains all of its safety properties, which will facilitate more precise conflict resolution in subsequent refinements. (see Figure.8).

```
MACHINE
        DetectConflic
REFINES
        M_Conflict
SEES
        C_Conflict
VARIABLES
.....
INVARIANTS
        inv1  :  Buffer ∈ DEVICE ↔ RESOURSE
        inv2  :  ConfType ∈ conflict_type → CONFLICT
```

Figure. 8. An Event-B DetectConflic Machine

- The ***refined Refine_ConflictDetect*** event retains the abstract objective of identifying the presence of unsafe devices while incorporating explicit guards. The *grd3* clarifies the reasons for conflicts in the abstract conflict scenario. One of the identified conflict types indicates actual conflict. The *grd4* assigns a category to the conflict, facilitating classification without altering the abstract behaviour. Then *act3* enhance the model by categorising each conflict according to a specific type that is not present in the abstract machine (see Figure.9).

```
Refine_ConflictDetect ≙ WHERE
.....
    grd3: (dev1 ∈ Active_dev ∧ dev2 ∈ Active_dev ∧ (dev1↦dev2)
          ∈ Fun_dev) ∨ (∃res· (dev1, res) ∈ Buffer ∧ (dev2, res) ∈
          Buffer) ∨ ((priority (dev2), priority (dev1)) ∈ prior_dev
          ∧ dev2 ∈ Active_dev ∧ dev1 ∈ pend
    grd4: confT ∈ CONFLICT ∧ confT = conflict_type (dev1, dev2)
THEN
   ........
    act3: conflict_type ≔ conflict_type ∪ {(dev1↦dev2) ↦ confT,
          (dev2↦dev1) ↦ confT}
END
```

Figure. 9. The Event Refine_ConflictDetect.

- The ***ClearBuffer*** event is a new event introduced in this refinement. This event eliminates resolved abnormal items once enough resource allocation has been recovered, ensuring that the buffer contains only relevant and current observations (see Figure.11).

```
ClearBuffer ≙ WHERE
    grd1: (dev, res) ∈ Buffer ∧ alloc_Res (dev, res)
          ≥ reso_req (dev, res)
    THEN
    act1: Buffer ≔ Buffer \ {(dev, res)}
END
```

Figure. 11. The Event ClearBuffer.

5.3 Second Refinement (Conflict Resolution Model)

This model is a second refinement of ours that enhances the conflict detection techniques established in the abstract model by incorporating proactive prevention features, rather than reacting to conflicts after they have occurred. This change actively prevents possible resource conflicts by using reservation tables and capacity budgets based on priority levels, instead of just reacting to problems when they occur. Resources are allocated according to priorities to guarantee that high-priority and crucial devices consistently have access to adequate capacity, hence, limiting resource allocation decisions prior to the emergence of disputes. This model satisfies the requirements *Prio1, Prio2* and *Prio3*.

- **Machine (ResolutionConflict):**

This machine is an extension of the DetectConflict machine, which establishes important safety and proactively avoids conflicts. The *inv1* is a natural number that reserves the priority level and resource type. In the *inv2*, the device can link the ownership capability to priority levels, making it easy to change priorities. Additionally, the *inv3* ensures the limiting of reserved resources to actual capacities. Lastly, to prevent resource conflicts and preserve system accuracy and usability, *inv4* aligns device requirements with budgets. See Figure 12.

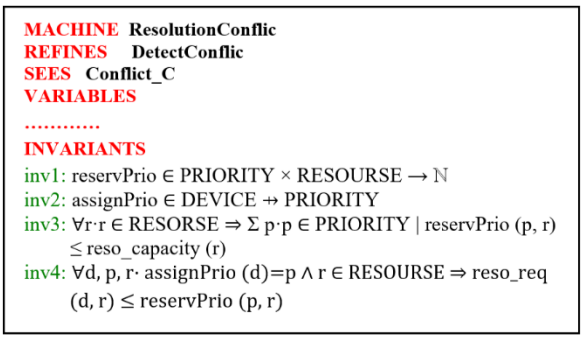

Figure.12. An Event-B ResolutionConflic Machine.

- The event named ***AssignPriority*** : A *dev* (parameter) and a Prio defined in this event set (*grd1*) are used to determine this event for assigning an efficient priority level set (*grd2*). This allows the system, as illustrated in Figure 13, to dynamically modify a device's priority while maintaining consistency with the initial priority ordering set (*act1*).

- The event ***ReservResource*** addresses the *Prior1* requirement by designating a reserved portion of a resource at a specific priority level. The system verifies the availability of the priority and resource set (*grd1*) and ensures that the reserved number of resources does not exceed the available capacity set (*grd2*). If valid, it modifies the reservation table, therefore avoiding conflicts and ensuring resources for each established priority level (*act1*). illustration in Figure 14.

- The ***Refine_PriorityAllocate*** event in the original *AllocatePriority* event, allocation was permitted if resource utilisation did not exceed its maximum capacity. The refined event limits device allocation to an effective priority level set for *grd2* and ensures that the assigned resource amount does not exceed the reserved budget for *grd3* and *grd4*. This extra guard provides proactive conflict avoidance by requiring priority-based resource budgeting prior to activation (See Figure. 15).

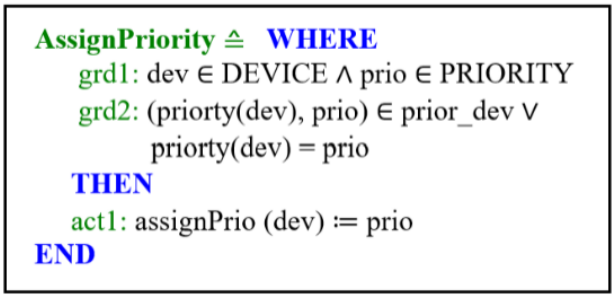

Figure. 13. The Event AssignPriority.

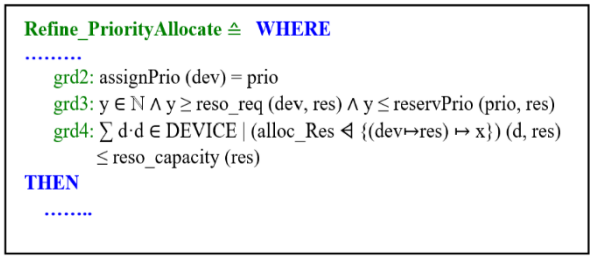

Figure. 14. The Event ReservResource).

ReservResource ≙ WHERE
grd1: prio ∈ PRIORITY ∧ res ∈ RESOURSE
∧ x ∈ ℕ
grd2: ∑ p·p ∈ PRIORITY | (reservPrio ⩤
{(prio↦res) ↦ x}) (p, res) ≤ reso_capacity (res)
THEN
act1: reservPrio ≔ reservPrio ⩤ {(prio↦res) ↦ x}
END

Figure. 15. The Event Refine_PriorityAllocate.

6. The proposed model verification and validation

This section outlines the procedures employed to verify and validate our model, our validation is made up of two parts:ProB[20]and Proof obligations (POs) [21].

6.1 Proof-based verification

The mathematical specification of the Event-B model enables the development of tests for model consistency with formal constraints with the assistance of proof obligations (POs) generated by the Rodin platform. Type consistency (verification of appropriate data types for variables), POs in Event-B must adhere to certain standards to be valid: invariant preservation, event compatibility, refinement consistency, and convergence. Invariants must remain valid before and after an event executes, and the system must meet all requirements for valid implementation at all enhancement levels [22]. Event-B model components use automated theorem provers to verify the correctness of the model and to specify system behaviour. To ensure safety and dependability, safety-critical systems need to have their behaviours thoroughly specified and verified [23].

Actually, demonstrating this theorem is required to ensure this model is correct. Proof problems were generally classified as well-definedness (WD), invariant preservation (INV) (for model invariants), or guard strengthening (GRD). The well-definedness proof obligation rule (WD) ensures that any possible ill-defined theorem, axiom, invariant, guard, action, witness, or variant actually is well defined. When referring to a specific modelling component, use the following element names: inv/WD, grd/WD, and act/WD. The application of this proof obligation rule is determined by the expression, which may or may not be evident. A PO can be discharged automatically or interactively (green symbol) or left undischarged (orange symbol). The discharge is automatically processed when the letter "A" appears on the PO [24].

In summary, 75 POs have been created. The automatic prover discharges 60 of them. The remaining proofs are insufficient not because they are difficult, but because they include several processes. Finally, we effectively guided the interactive prover to the relevant methods and guidelines, accelerating the completion of these proofs.

Robust Rodin plugins significantly simplify our work by automating most repetitive mechanical processes. However, careful modelling and more intelligent automatic proof tools can reduce the amount of time individuals must spend on verification duties. Figure 16 displays the proof statistics that show the distribution of purchase orders (POs) released manually and automatically for The IoT conflict resolution model.

Element name	Total	Auto.	Manual	Reviewed	Undischarged
IoT Conflict resolution	**75**	**60**	**15**	**0**	**0**
M_Conflict	33	28	5	0	0
C_Conflict	0	0	0	0	0
DetectConflict	17	13	4	0	0
ResolutionConflict	25	19	6	0	0

Figure. 16. The Event-B Statistical Proof of IoT Conflict Resolution Model.

Remember that the Event-B approach does not suffer from the state explosion problem that plagues other verification methods in the literature.

6.2 Validation by ProB model checker

The document discusses the use of the ProB animation tool as a plugin for Rodin [25]. ProB facilitates validation of requirements and error detection without user intervention for most proof obligations. It allows for model creation, deadlock checking, and test case generation, although it requires a concrete model rather than an abstract Event-B specification. We automated the verification of model specifications step-by-step using ProB, successfully demonstrating the animation of case studies through the manipulation of variables and constants in a specific model.

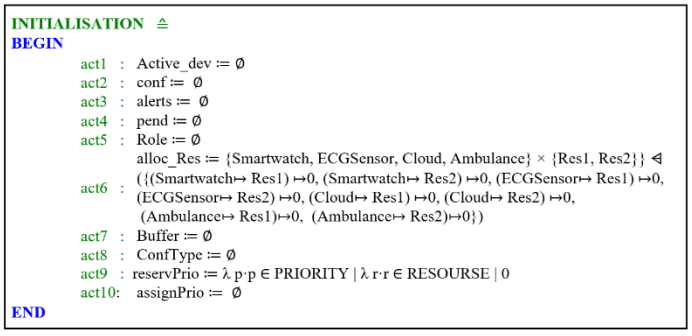

Figure. 17. The Event-B initialisation of IoT Conflict Resolution Model.

The suggested Event-B model is verified via a case study on ECG monitoring that includes a smartwatch, wearable ECG sensor, healthcare cloud, and ambulance [26]. Normal operation is defined by periodic data transmission between the smartwatch and a sensor without conflicts. Upon detection of abnormal ECG patterns, the system proactively discovers and resolves resource, functional, and priority issues through a priority-based mechanism. High-priority emergency services on the Healthcare Cloud supersede lower-priority tasks and initiate ambulance activation.

The ProB animation checks that services that can't work together don't run at the same time, makes sure important services have enough resources, and solves conflicts without getting stuck, showing how well the suggested proactive conflict resolution strategy works for safety-critical IoT healthcare systems. This analysis specifically concentrates on the initialisation parameters of the electrocardiogram (ECG) IoT system, as illustrated in Figure. 17. ProB's testing confirmed that the model's events exhibited no issues, indicating an effective fix to its behaviour.

The subsequent phases of the animation are as follows (See Figure. 18):

- We begin by starting the SETUP-CONTEXT event, which is responsible for setting the context's initial values for carrier sets and constants;
- Initiate the INITIALISATION event to reset the initial state;
- Lastly, proceed to validate the stages of the scenario. For each cycle's events, the animator determines which guards are true, starts the events with those guards, and shows the parameters that meet those guards. After an event occurs, the animator verifies that the invariants are still valid and computes replacements.

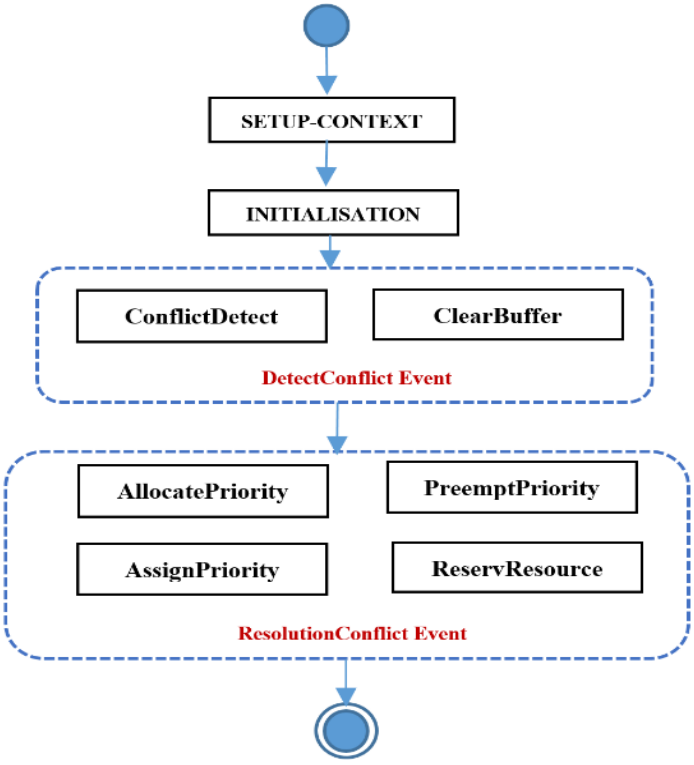

Figure. 17. The Statistical Proof of the Event-B IoT Conflict Resolution Model.

7. Conclusion

Since the functioning of large-scale IoT systems depends on trusted automation, maintaining a safe and secure operation is essential. The Event-B formal method is employed in this paper to introduce a novel approach for the detection and resolution of conflicts in IoT systems through the use of priority. We define and account for a set of requirements in our model. The constructed model comprises three sub-models: the first refined model (Conflict Detection), the second refined model (Conflict Resolution), and the abstract model. By progressively constructing all IoT conflict resolution systems and facilitating the proofs, the refinements enabled us to rectify the situation. In our model, an electrocardiogram (ECG) IoT system was implemented as a case study. Ultimately, we verify the correctness of our formal model using an animator checker tool, ProB, and proof obligations. Our approach thus provides more dependable guarantees and error-free solutions to the conflict in the IoT system. Future research will focus on enhancing scalability and adapting dynamic prioritization in large-scale IoT environments.

References

[1] Ryan, P.J., Watson, R.B.: Research challenges for the internet of things: what role can or play? Systems 5(1), 24 (2017)
[2] Va¨yrynen, R.: Regional conflict formations: an intractable problem of international relations. J. Peace Res. 21(4), 337–359 (1984)
[3] Mohammed, I. A. (2022). A comprehensive mapping research on the management of conflicts in IoT-based systems actuation toward DevOps.
[4] Pradeep, P., Kant, K.: Conflict detection and resolution in IoT systems: a survey. IoT 3(1), 191–218 (2022)
[5] Huang, B., Chen, C., Lam, K. Y., & Huang, F. (2024, July). Proactive Detection of Physical Inter-rule Vulnerabilities in IoT Services Using a Deep Learning Approach. In 2024 IEEE International Conference on Web Services (ICWS) (pp. 164-171). IEEE.
[6] Toman, S.H., Hamel, L., Toman, Z.H., Graiet, M., Ouchani, S.:Formal modelling and verification of scalable service composition in IoT environment. SOCA 17(3), 213–231 (2023)
[7] Abrial, J.R.: On b and Event-B: principles, success and challenges.In: Abstract State Machines, Alloy, B, TLA, VDM, and Z: 6th International Conference, ABZ 2018, Southampton, UK, June 5–8, 2018, Proceedings 6, pp. 31–35. Springer (2018)
[8] Wing,J. M.(1990).A specifier's introduction to formal methods.Computer,23(9),8-22.
[9] Toman, S.H., Hamel, L., Graiet, M.: Refinement and verification for IoT service composition. In: 2023 IEEE Symposium on Computers and Communications (ISCC), pp. 483–486 (2023). https://doi.org/10.1109/ISCC58397.2023.10218287
[10] Toman, Z.H., Hamel, L., Toman, S.H., Graiet, M., Valadares, D.C.G.: Formal verification for security and attacks in IoT physical layer. J. Reliab. Intell. Environ. 2021, 1–19 (2023)
[11] Toman, S.H., Lahouij, A., Hamel, L., Toman, Z.H., Graiet, M.: A correct by construction model for cbps systems verification. In: 2023 IEEE Symposium on Computers and Communications (ISCC). pp. 1299–1304 (2023). https://doi.org/10.1109/ISCC58397.2023.10217842

[12] Toman, S.H., Lahouij, A., Kotel, S., Hamel, L., Toman, Z.H.,Graiet, M.: Service to service communication based on CBPS system: refinement and verification. Soft. Comput. 2024, 1–21 (2024)
[13] A. Al Farooq, E.Al-Shaer,T.Moyer,and K. Kant,"IoTC2:A formal method approach for detecting conflicts in large scale iot systems",In 2019IFIP/IEEE symposium on integrated network and service management(IM),pp 442-447,2019.
[14] 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]
[15] Shehata, M.; Eberlein, A.; Fapojuwo, A.O. A taxonomy for identifying requirement interactions in software systems. Comput.Netw. 2007, 51, 398–425. [CrossRef]
[16] 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]
[17] 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.
[18] 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.
[19] 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.
[20] Hoang,T.S.,"An introduction to the Event-B modelling method," Industrial Deployment of System Engineering Methods,2013,pp.211-236.
[21] M.Leuschel,and M.Butler,"ProB:A model checker for B,"In International symposium of formal methods europe(pp.855-874). Springer,Berlin,Heidelberg,2013.
[22] Hallerstede, S.: On the purpose of Event-B proof obligations. In:Abstract State Machines, B and Z: First International Conference,ABZ 2008, London, UK, September 16-18, 2008, Proceedings 1,pp. 125–138. Springer (2008)
[23] Abrial, J.R.: From z to b and then Event-B: assigning proofs to meaningful programs. In: Integrated Formal Methods: 10th International Conference, IFM 2013, Turku, Finland, June 10-14,2013, Proceedings 10, pp. 1–15. Springer (2013)
[24] Abrial, J.: The B-book Assigning Programs to Meanings. Cambridge University Press, Cambridge (2005)
[25] Leuschel M, Butler M (2013) ProB: A model checker for B.In: International symposium of formal methods Europe.Springer,Berlin, Heidelberg, pp 855–874
[26] Muankid A, Ketcham M (2019) The real-time electrocardiogram signal monitoring system in wireless sensor network. Int J Online Biomed Eng 15(2)
[27] Pradeep, P., & Kant, K. (2022). Conflict detection and resolution in IoT systems: a survey. IoT, 3(1), 191-218.
[28] Huang, B., Dong, H., Bouguettaya, A.: Conflict detection in IoTbased smart homes. In: 2021 IEEE International Conference on Web Services (ICWS), pp. 303–313. IEEE (2021)
[29] Jover, J., Bermúdez, A., & Casado, R. (2022). Priority-aware conflict resolution for U-space. Electronics, 11(8), 1225.