International Journal for Multidisciplinary Research (IJFMR)
E-ISSN: 2582-2160 ● Website: www.ijfmr.com ● Email: editor@ijfmr.com

Smart Trigger Execution Framework in IoT Gateway Applications for Home Automation

Omkar Wagle[1], Abhijeet Sonar[2], Aishwarya Lonarkar[3]

[1]ov.wagle@gmail.com, Independent Researcher, CA
[2]abhijeetsonar.us@gmail.com, Independent Researcher, India
[3]aishwaryalonarkar@gmail.com, Independent Researcher, TX

Abstract

This paper presents a trigger-based execution framework designed to enhance the responsiveness, flexibility, and reliability of Zigbee-based home automation systems. The framework enables users to define automation triggers—including conditions and corresponding actions-via a mobile application, which are then centrally managed by a local gateway. The gateway processes incoming data from Zigbee sensors in real time, evaluates trigger conditions, and executes actions by interfacing with Zigbee[1] devices. A relational database schema supports structured storage of triggers, conditions, actions, and device metadata. Additionally, the integration of the Paho MQTT [2] client within the Zigbee[1] System-on-Chip (SoC) application allows for direct MQTT[2] communication, improving system autonomy and reducing latency. This architecture minimizes cloud dependency, enhances data privacy, and facilitates dynamic device control, making it a robust and future- ready solution for smart home automation. The proposed system is particularly suited for user-centric environments where intelligent, local decision-making is critical.

Keywords: Paho MQTT, Zigbee, Smart Trigger, Home automation, Internet of Things (IoT)

1. Introduction

The evolution of smart home technologies has led to a growing demand for automation systems that are intelligent, responsive, and user-friendly. These systems aim to simplify daily life by allowing users to automate device behaviors based on environmental inputs such as triggering lights when motion is detected or adjusting heating systems in response to temperature changes. Achieving such functionality requires a robust and flexible framework capable of defining, storing, and executing automation logic efficiently and securely.

This paper introduces a trigger-based execution framework tailored for Zigbee-based home automation environments. At the core of the system is a mobile application that allows users to define automation triggers, each composed of conditions (e.g., sensor thresholds) and corresponding actions (e.g., device commands). These triggers are stored locally in a structured database on the gateway, which also serves as the central decision-making application. By offloading trigger evaluation and action execution to the gateway, the system minimizes reliance on cloud services, thereby improving latency, enhancing privacy, and ensuring greater operational reliability.

Furthermore, the integration of the Paho MQTT[2] client into the Zigbee System-on-Chip (SoC)

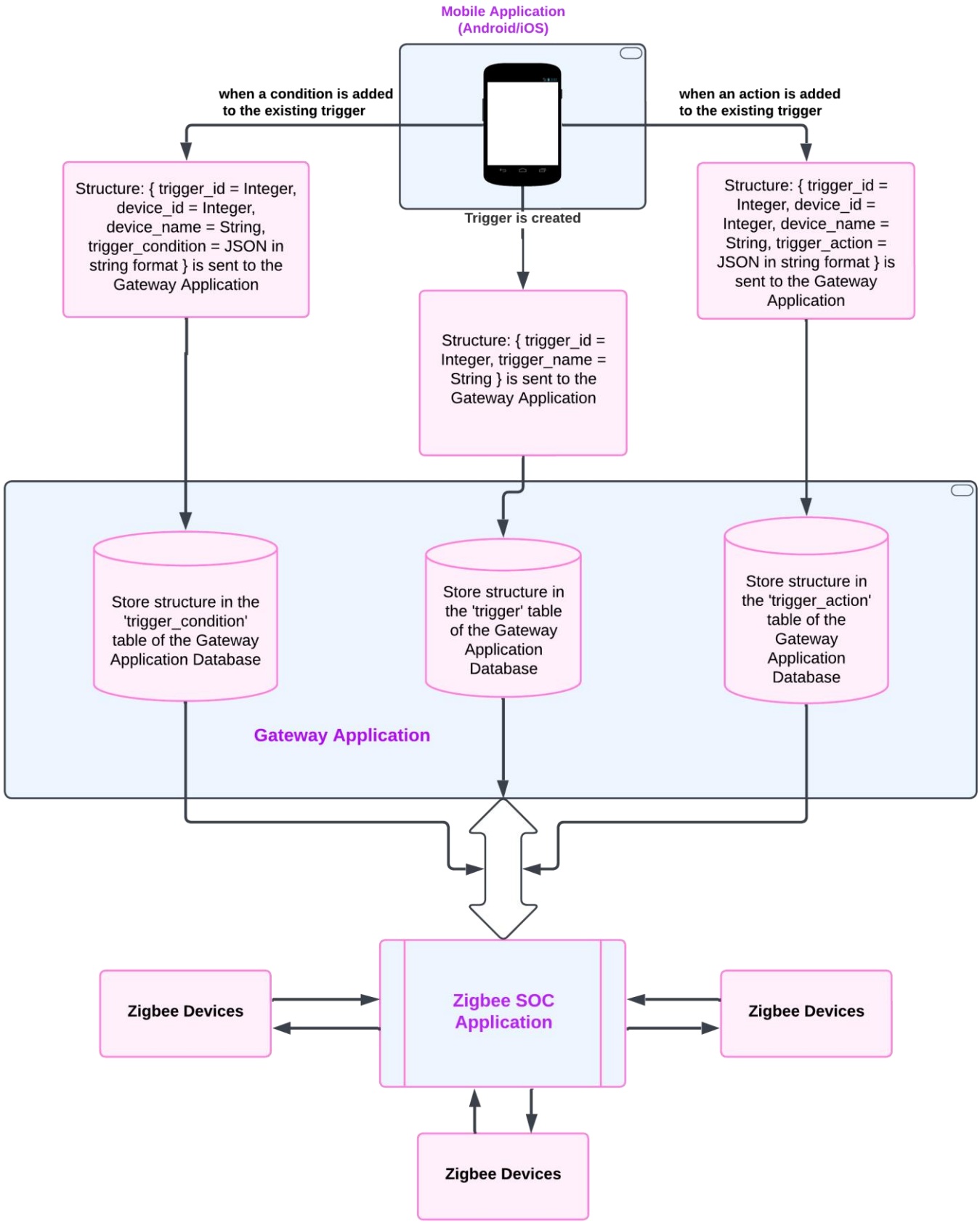
application enables direct, bidirectional communication between Zigbee devices and the MQTT[2] broker. This design reduces the burden on the gateway for handling message routing, supports real-time device updates, and improves the system's modularity and scalability. The resulting architecture offers a responsive, secure, and extensible platform for smart home automation, capable of adapting to a wide range of Internet of Things (IoT)[3] scenarios.

2. System Architecture Overview

The Mobile Application (Android/iOS) serves as the user interface for creating and managing automation triggers involving Zigbee-based devices. When a user wants to automate a task, they start by creating a trigger, which is identified by a unique trigger_id and given a user-friendly trigger_name. This trigger information is then sent to the Gateway Application and stored in the local database under the trigger table. Once a trigger is created, the user can add one or more conditions to it. Each condition links a specific sensor device (e.g., a temperature or motion sensor) to the trigger and defines a logic trigger in JSON format—such as "temperature < 15°C." The condition structure includes the trigger_id, device_id, device_name, and the trigger_condition as a JSON string. This structure is also sent to the gateway, where it is stored in the trigger_condition table. Following the condition, the user can specify an action that should be performed when the condition is satisfied.

For example, actions like turning on a Zigbee bulb or locking a smart door are defined using trigger details and stored in the trigger_action table. The Gateway Application receives these settings from the mobile app and saves them in its SQLite[5] database. Zigbee sensors send real-time data to the gateway, which checks if any condition matches. If so, it fetches the corresponding action and sends it to the device via the Zigbee SoC, enabling smooth and automated device control based on sensor input.

This setup ensures devices react instantly without needing constant cloud access. It also gives users more privacy and control over their smart home environment.


Mobile Application
(Android/iOS)
when a condition is added
to the existing trigger
when an action is added
to the existing trigger
Structure: { trigger_id = Integer, device_id = Integer, device_name = String, trigger_condition = JSON in string format } is sent to the Gateway Application
Trigger is created
Structure: { trigger_id = Integer, device_id = Integer, device_name = String, trigger_action = JSON in string format } is sent to the Gateway Application
Structure: { trigger_id = Integer, trigger_name = String } is sent to the Gateway Application
Store structure in the 'trigger_condition' table of the Gateway Application Database
Store structure in the 'trigger' table of the Gateway Application Database
Store structure in the 'trigger_action' table of the Gateway Application Database
Gateway Application
Zigbee Devices
Zigbee SOC Application
Zigbee Devices
Zigbee Devices

3. Mobile Application:

3.1. Trigger Creation via Mobile Application

The automation process begins when a user opens the mobile application, which serves as the primary interface for configuring smart device behaviors. Within the app, the user initiates the creation of a new automation trigger, known as a "trigger." This trigger acts as a blueprint for the automation, defining the conditions under which specific actions should occur.

Upon creating a trigger, the system assigns it a unique identifier, referred to as trigger_id, and allows the user to provide a descriptive name, trigger_name, for easy reference. This information is encapsulated in a structured format:

- **Trigger Structure**: {trigger_id: Integer, trigger_name: String}

This structured data is then transmitted to the Gateway Application, where it is stored in the trigger table of the local SQLite[5] database. This storage ensures that the system can efficiently manage and reference the trigger during operation.

3.2. Trigger Condition Creation via Mobile Application

After establishing a trigger, the user can define specific conditions that determine when the trigger should activate. These conditions are linked to particular devices within the system. For instance, a user might set a condition to monitor the temperature reported by a sensor. To add a condition, the user selects the relevant device and specifies the condition in a JSON- formatted string. This structured data includes the trigger's identifier, the device's identifier and name, and the condition itself:

- **Condition Structure**: {trigger_id: Integer, device_id: Integer, device_name: String, trigger_condition: JSON String}

This condition structure is sent to the Gateway Application and stored in the trigger_condition table of the SQLite[5] database. By organizing conditions in this manner, the system can efficiently evaluate whether the specified criteria are met during operation.

3.3. Trigger Action Creation via Mobile Application

Once conditions are set, the user defines the actions that should occur when those conditions are satisfied. Actions are also associated with specific devices. For example, if a temperature sensor detects a reading below a certain threshold, the system might turn on a heater. To specify an action, the user selects the target device and defines the desired action in a JSON-formatted string. The action structure includes the trigger's identifier, the device's identifier and name, and the action details:

- **Action Structure**: {trigger_id: Integer, device_id: Integer, device_name: String, trigger_action: JSON String}

This action structure is transmitted to the Gateway Application and stored in the trigger_action table of the SQLite[5] database. Storing actions in this structured format allows the system to execute the appropriate responses when conditions are met.

4. Gateway Application

In an IoT automation framework, the gateway serves as the central hub that orchestrates the interaction between the user- defined automation triggers and the connected devices. The gateway receives this information when a user creates a new trigger through the mobile application. It stores it in the trigger table of its local SQLite[5] database. Each trigger is assigned a unique identifier (trigger_id), which is the unique identifier in the trigger table.

Subsequently, when the user defines specific conditions and actions associated with this trigger, the gateway stores these details in the trigger_condition and trigger_action tables, respectively. Both tables reference the trigger_id as a foreign key, establishing a relational link to the original trigger. This structured approach ensures that each condition and action is accurately associated with its corresponding trigger, facilitating efficient retrieval and execution.

In addition to managing triggers, the gateway maintains a complete record of all devices linked to the network in

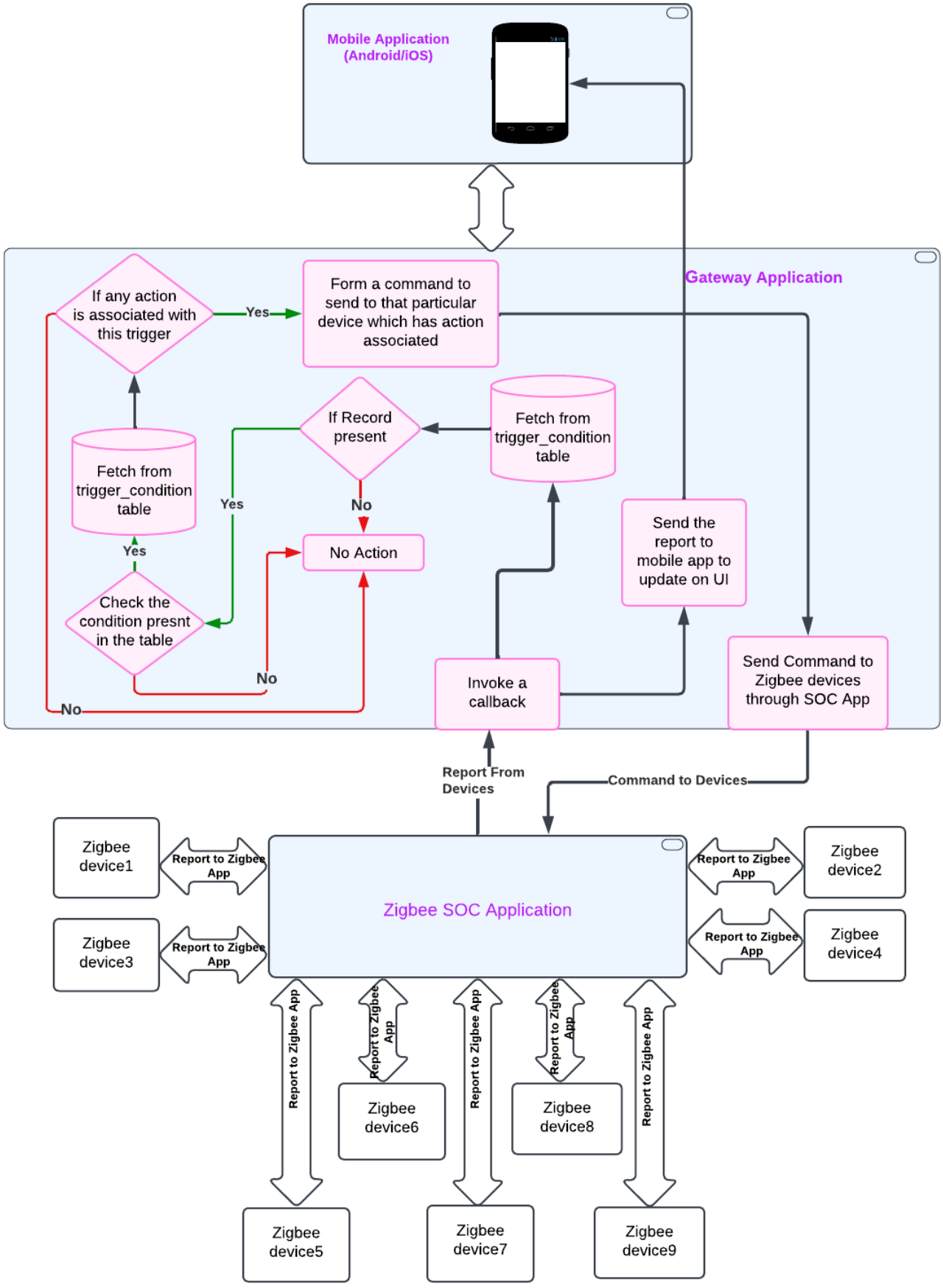
the device_info table. This table includes critical information such as device_id, device_name, manufacturer details, device capabilities, and attributes. By referencing this table, the system ensures that only recognized and registered devices are involved in automation triggers, thereby maintaining system integrity and preventing unauthorized device interactions.

The gateway continuously monitors incoming data from Zigbee devices through the Zigbee System-on-Chip (SoC) application. Upon receiving a report from a device, the gateway initiates a multi-step process:

4.1. ***Condition Matching***: The gateway queries the trigger_condition table using the device_id and device_name from the incoming report to identify any matching conditions.

4.2. ***Evaluation***: If matching conditions are found, the gateway iterates through each, evaluating whether the current state or data from the device satisfies the specified condition.

4.3. ***Action Retrieval***: For each condition that evaluates to true, the gateway retrieves the corresponding action from the trigger_action table using the associated trigger_id.


Mobile Application (Android/iOS)
Gateway Application
If any action is associated with this trigger
Yes
Form a command to send to that particular device which has action associated
Fetch from trigger_condition table
If Record present
Fetch from trigger_condition table
Yes
No
Yes
No Action
Check the condition presnt in the table
No
No
Send the report to mobile app to update on UI
Invoke a callback
Send Command to Zigbee devices through SOC App
Report From Devices
Command to Devices
Zigbee SOC Application
Zigbee device1
Zigbee device3
Zigbee device2
Zigbee device4
Zigbee device5
Zigbee device6
Zigbee device7
Zigbee device8
Zigbee device9
Report to Zigbee App

4.4. ***Command Execution***: The gateway constructs a command payload based on the retrieved action details, incorporating the relevant device_id and device_name. This command is then dispatched to the appropriate Zigbee device to execute the desired action.

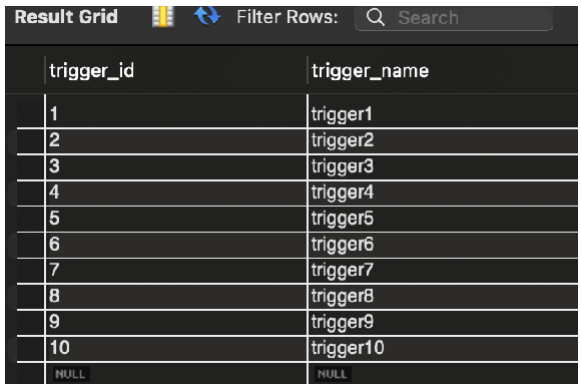
5. Database Schema and Operational Workflow

The Gateway Application employs a structured relational database to manage automation triggers effectively. This database comprises several interconnected tables, each serving a specific purpose in the automation process.

5.1. *trigger Table*

This table serves as the foundational registry for all automation triggers defined by users. Each entry in the trigger table includes:

- trigger_id: A unique integer that acts as the primary key, ensuring each trigger can be distinctly identified.
- trigger_name: A descriptive name the user provides to reference the trigger easily.



trigger_id	trigger_name
1	trigger1
2	trigger2
3	trigger3
4	trigger4
5	trigger5
6	trigger6
7	trigger7
8	trigger8
9	trigger9
10	trigger10
NULL	NULL

The trigger_id is crucial as it establishes relationships with other tables, linking specific conditions and actions to their corresponding triggers.

5.2. *trigger_condition Table*

This table records the specific conditions under which a trigger should activate. Each condition is associated with a particular device and includes:

- trigger_id: A foreign key referencing the trigger table, indicating which trigger the condition belongs to.
- device_id: An identifier for the device involved in the condition.
- device_name: The name of the device, providing a human-readable reference.
- trigger_condition: A JSON-formatted string detailing the specific condition (e.g., temperature < 15°C).

Result Grid Filter Rows: Search Export:

trigger_id	device_id	device_name	trigger_condition
1	1001	Temp Sensor	{"parameter": "temperature", "operator": "<", "value": 15}
2	1002	Motion Sensor	{"parameter": "motion", "operator": "==", "value": "detected"}
3	1003	Light Sensor	{"parameter": "light", "operator": "<", "value": 300}
4	1004	Humidity Sensor	{"parameter": "humidity", "operator": ">", "value": 70}
5	1005	CO2 Sensor	{"parameter": "co2", "operator": ">", "value": 1000}
6	1006	Door Sensor	{"parameter": "door", "operator": "==", "value": "open"}
7	1007	Water Leak Sensor	{"parameter": "leak", "operator": "==", "value": "true"}
8	1008	Gas Sensor	{"parameter": "gas", "operator": ">", "value": 400}
9	1009	Smoke Sensor	{"parameter": "smoke", "operator": "==", "value": "present"}
10	1010	Sound Sensor	{"parameter": "sound", "operator": ">", "value": 70}

By structuring conditions in this manner, the system can efficiently evaluate whether incoming data from devices satisfies any user-defined conditions.

5.3. trigger_action Table

This table defines the actions to be executed when corresponding conditions are met. Each action entry includes:

- trigger_id: A foreign key linking back to the trigger table, indicating which trigger the action is associated with.
- device_id: The identifier of the device that will perform the action.
- device_name: The name of the device, aiding in identification.
- trigger_action: A JSON-formatted string specifying the action to be taken (e.g., turn on a smart bulb).

Result Grid Filter Rows: Search Export:

trigger_id	device_id	device_name	trigger_action
1	2001	Zigbee Bulb	{"action": "switch_on", "value": true}
2	2002	Ceiling Fan	{"action": "set_speed", "level": 3}
3	2003	Smart Plug	{"action": "turn_off"}
4	2004	Heater	{"action": "switch_on", "duration": 600}
5	2005	Air Purifier	{"action": "increase_fan_speed", "level": 2}
6	2006	Door Lock	{"action": "lock"}
7	2007	Water Valve	{"action": "close_valve"}
8	2008	Gas Alarm	{"action": "activate_alarm"}
9	2009	Exhaust Fan	{"action": "switch_on"}
10	2010	Speaker	{"action": "play_alert", "sound": "buzzer"}

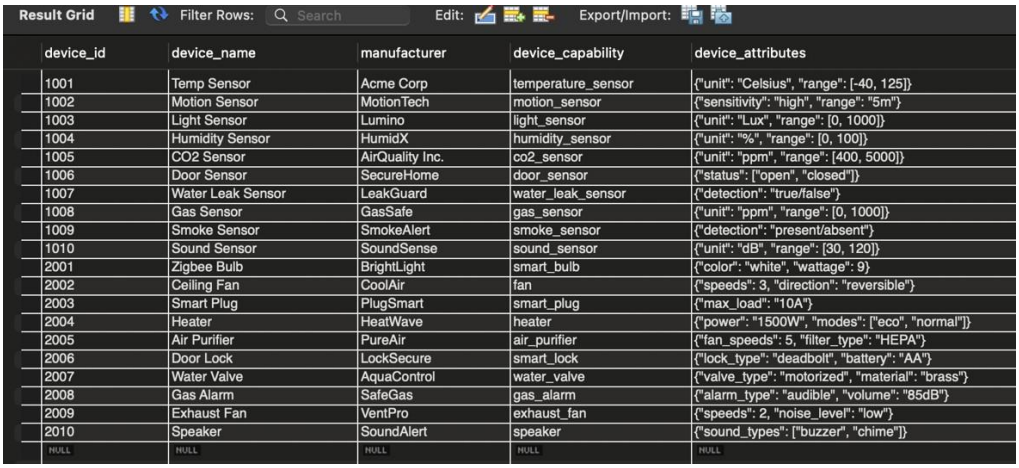
This structure allows the system to execute precise actions on designated devices when conditions are fulfilled.

5.4. device_info Table

To manage and validate devices within the network, the device_info table maintains comprehensive records of all connected devices, including:

- device_id: A specific identifier that differentiates one device from another.
- device_name: The human-readable name of the device.
- manufacturer: Information about the device's manufacturer.

- device_capability: Details about what the device can do (e.g., sensing temperature, controlling lights).
- device_attributes: Additional properties or settings relevant to the device.



Result Grid | Filter Rows: Search | Edit: | Export/Import:

device_id	device_name	manufacturer	device_capability	device_attributes
1001	Temp Sensor	Acme Corp	temperature_sensor	{"unit": "Celsius", "range": [-40, 125]}
1002	Motion Sensor	MotionTech	motion_sensor	{"sensitivity": "high", "range": "5m"}
1003	Light Sensor	Lumino	light_sensor	{"unit": "Lux", "range": [0, 1000]}
1004	Humidity Sensor	HumidX	humidity_sensor	{"unit": "%", "range": [0, 100]}
1005	CO2 Sensor	AirQuality Inc.	co2_sensor	{"unit": "ppm", "range": [400, 5000]}
1006	Door Sensor	SecureHome	door_sensor	{"status": ["open", "closed"]}
1007	Water Leak Sensor	LeakGuard	water_leak_sensor	{"detection": "true/false"}
1008	Gas Sensor	GasSafe	gas_sensor	{"unit": "ppm", "range": [0, 1000]}
1009	Smoke Sensor	SmokeAlert	smoke_sensor	{"detection": "present/absent"}
1010	Sound Sensor	SoundSense	sound_sensor	{"unit": "dB", "range": [30, 120]}
2001	Zigbee Bulb	BrightLight	smart_bulb	{"color": "white", "wattage": 9}
2002	Ceiling Fan	CoolAir	fan	{"speeds": 3, "direction": "reversible"}
2003	Smart Plug	PlugSmart	smart_plug	{"max_load": "10A"}
2004	Heater	HeatWave	heater	{"power": "1500W", "modes": ["eco", "normal"]}
2005	Air Purifier	PureAir	air_purifier	{"fan_speeds": 5, "filter_type": "HEPA"}
2006	Door Lock	LockSecure	smart_lock	{"lock_type": "deadbolt", "battery": "AA"}
2007	Water Valve	AquaControl	water_valve	{"valve_type": "motorized", "material": "brass"}
2008	Gas Alarm	SafeGas	gas_alarm	{"alarm_type": "audible", "volume": "85dB"}
2009	Exhaust Fan	VentPro	exhaust_fan	{"speeds": 2, "noise_level": "low"}
2010	Speaker	SoundAlert	speaker	{"sound_types": ["buzzer", "chime"]}
NULL	NULL	NULL	NULL	NULL

This table ensures that only recognized and properly configured devices are involved in automation triggers, maintaining system integrity.

6. Zigbee SOC Application

6.1. Operational Workflow of the Zigbee SOC application

6.1.1. Device Communication

All Zigbee devices in the network connect through a central component called the Zigbee SOC application. This application acts like the brain of the network's communication system. It manages how devices talk to each other by following specific triggers (called communication protocols). Its job is to make sure messages sent between devices are delivered correctly and reliably, preventing data loss or errors.

6.1.2. Data Reception

Whenever a Zigbee device (such as a sensor or switch) wants to share information-like sending a status update or reporting sensor data-it sends this information to the network. The Zigbee SOC application receives this incoming data and acts as a middleman by passing it along to another software called the gateway application. This gateway is responsible for understanding and processing the data to decide what needs to be done next.

6.1.3. Command Transmission

After the gateway application processes the data from the devices, it might decide that an action should be taken. For example, if a temperature sensor reports that the room is too hot, the gateway might decide to turn on a fan or AC unit. The gateway sends this instruction back to the Zigbee SOC application in the form of a command, specifying exactly what the device should do.

6.1.4. Device Control

Once the Zigbee SOC application receives the command from the gateway, it sends the command directly to the targeted Zigbee device (e.g., a smart light or motor). The device then performs the action requested, such as turning on, off, or adjusting settings. This completes the communication

loop, enabling real-time control and automation within the network.

In modern Zigbee-based home automation systems, integrating the Paho MQTT[2] client directly into the Zigbee SOC application streamlines communication between Zigbee devices and the MQTT broker. This architecture allows the Zigbee SOC to handle MQTT[2] messaging autonomously, reducing dependency on intermediary gateway applications. The Zigbee SOC, equipped with a System-on-Chip (SoC), interfaces with various Zigbee devices such as sensors and actuators. By incorporating the Paho MQTT[2] client, the SOC can publish sensor data to specific MQTT[2] topics (e.g., zigbee2mqtt/device_name) and subscribe to command topics (e.g., zigbee2mqtt/device_name/set). This bidirectional communication enables real-time monitoring and control of devices through the MQTT[2] broker.

Implementing the Paho MQTT[2] client on the Zigbee SOC offers several advantages:

- **Efficiency**: Direct MQTT[2] communication reduces latency and overhead associated with routing messages through a separate gateway application.
- **Scalability**: The publish/subscribe model of MQTT[2] facilitates easy addition of new devices and services without significant changes to the existing infrastructure.
- **Reliability**: Features like automatic reconnection and Quality of Service (QoS) levels in the Paho MQTT[2] client enhance the robustness of the communication, ensuring message delivery even in unstable network conditions.

  This setup is particularly beneficial in scenarios requiring prompt response times and minimal system complexity. By offloading MQTT[2] responsibilities to the Zigbee SOC, the system achieves a more modular and maintainable architecture, conducive to the dynamic needs of home automation environments.

7. Conclusion

The proposed trigger execution framework delivers a comprehensive, scalable, and efficient solution for managing intelligent behaviors in Zigbee-based home automation systems. The system ensures low-latency operation and enhanced privacy by enabling users to define automation triggers through a mobile application and offloading trigger processing to a local gateway, the system ensures low-latency operation and enhanced privacy. The modular database schema-including tables for triggers, conditions, actions, and device metadata-supports structured trigger storage and reliable execution. Integration of the Zigbee SoC with the Paho MQTT[2] client further enhances system responsiveness and autonomy by streamlining communication with the MQTT[2] broker. This architecture reduces reliance on cloud services and allows for real-time decision- making and secure device control, making it well-suited for dynamic, user-centric smart home environments. As the demand for responsive and intelligent home automation systems grows, the presented framework offers a future-ready foundation adaptable to a wide range of IoT scenarios.

8. References

1. Zigbee Alliance, *Zigbee Specification*, Zigbee Alliance, 2017. [Online]. Available: https://csa-iot.org/all-solutions/zigbee/
2. Eclipse Foundation, "Eclipse Paho - MQTT client library," [Online]. Available:

https://www.eclipse.org/paho/

3. J. Gubbi, R. Buyya, S. Marusic, and M. Palaniswami, "Internet of Things (IoT): A vision, architectural elements, and future directions," *Future Generation Computer Systems*, vol. 29, no. 7, pp. 1645–1660, 2013.
4. International Telecommunication Union, *Overview of the Internet of Things*, Recommendation ITU-T Y.2060, 2012. [Online]. Available: https://www.itu.int/rec/T-REC-Y.2060-201206-I/en.
5. D. R. Hipp, *SQLite*, SQLite Consortium. [Online]. Available: https://www.sqlite.org/