Hybrid IoT Platform Architectures for Flexible and Longstanding Deployments

Katarina Vuknić
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
katarina.vuknic@fer.unizg.hr

Mario Kušek
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
mario.kusek@fer.unizg.hr

Ivana Podnar Žarko
University of Zagreb, Faculty of Electrical Engineering and Computing
Zagreb, Croatia
ivana.podnar@fer.unizg.hr

Abstract

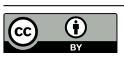
The paper analyzes hybrid IoT platform architectures that integrate cloud-based and edge-based solutions. While cloud platforms offer scalability and a centralized global view, edge platforms provide low latency, enhanced privacy, and local autonomy. By combining them, hybrid systems can leverage both local performance aspects (e.g., real-time actuation and device management) and global centralized processing at the cloud level. The edge is enabled to share filtered and aggregated local views with the cloud, while local changes of IoT devices at the edge level do not affect the cloud level. Such architectures are thus adequate for longstanding IoT deployments. We review existing strategies for integrating IoT devices with cloud platforms and introduce a flexible, edge-centric integration approach which simplifies device replacement procedures and offers configurable data synchronization with the cloud. We validate the approach through a case study on a hybrid smart home solution that integrates an open-source edge platform with a commercial cloud-based platform to manage and process home automation data across large-scale deployments.

Keywords

interoperability, intermediary edge gateway, IoT platform federation

1 Introduction

The integration of local and cloud platforms aims to combine the strengths of edge and cloud computing to achieve a more resilient, efficient and flexible system. Cloud platforms typically support application-layer communication protocols, such as MQTT (Message Queuing Telemetry Transport), CoAP (Constrained Application Protocol), or HTTP (HyperText Transfer Protocol). However, they often lack native support for device-level communication standards commonly used in home automation, industrial environments, or field monitoring, such as Zigbee, LoRa (Long Range), or BLE (Bluetooth Low Energy), which operate primarily on the link and physical layers of the communication stack. Consequently, most cloud-based IoT platforms do not provide direct drivers or plugins for such non-IP devices. In contrast, edge-based platforms, often open-source (e.g., Home Assistant [6], openHAB [22]), provide integration plugins for various low-level or lightweight protocols and device standards.



Within hybrid local–cloud architectures, these edge platforms or intermediate gateways can perform protocol translation, converting heterogeneous device messages into standardized data formats such as MQTT payloads or RESTful JSON objects. Such standardized representations can then be transmitted to the cloud for further processing including analytics, visualization, machine learning (ML), or digital twin synchronization. Collecting and pre-processing data locally before forwarding it to the cloud significantly reduces latency compared to cloud-only architectures, while also enhancing privacy by keeping raw data closer to the user. This design approach allows end users to benefit from the real-time-like responsiveness on the edge and operational continuity provided by the edge-based platform, as well as from the centralized management and advanced data analytics capabilities of the cloud platform. In cases where the edge and cloud components are developed by different vendors, an edge gateway layer should be introduced. This gateway acts as a protocol and data format translator, ensuring syntactic, semantic, and organizational interoperability. Firstly, with such setup, edge-based platform ensures continuous service to the user even when the connection to the cloud is temporarily unavailable. Secondly, data collected at the edge-based platform can be filtered before being sent to the cloud, reducing the processing load on the cloud and enhancing user privacy, as some data remains only locally. Thirdly, certain rules and automation can be executed locally for quick response, while more complex or coordinated operations can be managed from the cloud. For example, a washing machine connected to the edge-based platform may have a local rule to operate when electricity is low-cost, while the cloud can aggregate data from multiple households and determine an optimal distribution of appliance usage across a neighborhood, ensuring that not all machines start simultaneously at peak low-cost hours. Finally, the architecture should guarantee longevity and flexibility, enabling the replacement or migration of cloud providers without disrupting the overall system.

2 Related work

The manner in which an IoT platform obtains the state of end devices can be implemented at multiple levels, depending on the protocols and system architecture. In some cases, the device itself sends a notification of a change as soon as it occurs (push model), which allows for a fast response and lower latency. Alternatively, the platform can periodically poll the device to retrieve the current state (polling model), which is simpler to implement but introduces latency and additional network load [28]. The second dimension refers to where the communication takes place: within the local

https://doi.org/10.34749/3061-1008.2025.2

network, where the platform can manage devices even in the event of a connection failure, or via the manufacturer's cloud, where the platform depends on the availability of an external service. Combining these characteristics, we obtain four typical patterns of device-platform integration: local data push, local state polling, cloud notifications, and cloud data retrieval. This categorization [19] [21] is valid regardless of the specific platform and can also be applied to open-source solutions such as Home Assistant and openHAB [29].

Recent advances in IoT platform architectures have emphasized the importance of hybrid edge-cloud computing approaches that address the limitations of purely cloud-based solutions [2, 3]. The integration of edge computing with IoT devices has facilitated the creation of distributed computing architectures capable of handling massive data volumes generated by interconnected devices [30]. This convergence enables efficient data aggregation, analysis, and decision-making at the network's edge, reducing the burden on centralized cloud infrastructure and optimizing resource utilization [14].

2.1 Semantic Interoperability and Platform Integration

Generic platforms typically lack formal information models, allowing flexibility in connecting devices and defining data formats and protocols. This is especially convenient for dealing with a wide range of vendor-specific devices and enabling unified control and automation within a heterogeneous smart environment. The challenge of semantic interoperability has become increasingly important as IoT ecosystems expand beyond individual vendor silos. Recent research has focused on developing semantic interoperability support systems (SISS) that enable the generation of gateways or translator components between disparate data models [12, 13].

The FIWARE ecosystem represents a significant advancement in standardizing IoT platform interoperability through its NGSI-LD information model and API [9, 10]. FIWARE provides open-source components, called generic enablers, for building IoT platforms based on formal and de-facto standards. The NGSI-LD standard, formalized by the ETSI Industry Specification Group on Context Information Management, forms the basis of interoperability between core FIWARE components [9, 33].

Beyond the integration of individual end devices, we consider the integration of one IoT platform with another, where a local IoT platform (e.g., Home Assistant or openHAB) serves as a bridge to the cloud. An example of this approach is the connection between Home Assistant and the IBM Watson IoT Platform [8], for which Home Assistant provides an official integration. Through this integration, the platform registers directly as a gateway on Watson IoT and exchanges telemetry data and control commands via the MQTT protocol, without requiring any additional middleware. Also, there are similar integrations with Google Cloud [7] and Amazon Web Services [4], or modular cloud components such as Azure [5].

Recent studies have demonstrated the practical implementation of such integrations in real-world deployments. For instance, research on smart home automation systems using Home Assistant has shown significant improvements in energy efficiency, with systems achieving up to 22.5% reduction in overall energy consumption [11]. These implementations utilize various communication protocols including Zigbee, Z-Wave, and Wi-Fi to ensure seamless device interoperability [26, 29].

2.2 Cross-Platform Federation and Middleware Solutions

On the other hand, most cloud platforms are not natively supported by local IoT platforms, which necessitates intermediary solutions. A solution like SymbIoTe [24, 25], a platform-agnostic interoperability framework, works at a higher, cross-platform level, where different IoT platforms are connected via a common semantic model and standardized APIs. The SymbIoTe approach offers mediation services for search and controlled access to IoT resources across platforms in a uniform way, providing an IoT Portal with registration and search capabilities using semantic web technologies [25].

This makes it possible for the resources of one platform to be available to another, without the latter having to know internal protocols or implementations. In contrast, an approach utilizing edge gateway, e.g. Node-RED acting as a gateway at the local side [1, 17], operates at the local level, where it translates device protocols (SISS) and enables them to connect to the cloud or a single target platform. This is often simpler and more practical in real implementations, as it enables fast device integration and reliable management of local resources without requiring complex semantic models.

Recent research has emphasized the importance of publish/subscribe broker clustering for large-scale IoT applications, proposing hierarchical edge-cloud models that use efficient two-tier routing schemes to alleviate latency and scalability issues [23]. These approaches leverage proximate edge brokers strategically deployed in edge networks for data delivery services, demonstrating the effectiveness of distributed architectures in handling geo-distributed IoT devices.

The integration challenges extend beyond technical interoperability to include organizational and security considerations. Recent studies have highlighted the importance of secure semantic interoperability, particularly concerning the privacy implications of linking data across different IoT platforms [18]. These security considerations become increasingly critical as IoT deployments scale and involve multiple stakeholders across different domains.

3 Connecting devices to the cloud-based platform

There are four ways in which end devices can be connected to the IoT platform hosted in the cloud, either directly or via an edge-based platform acting as middleware, and with the varying components in between.

3.1 Direct (custom) integration

Directly integrating the end device to the platform in the cloud doesn't require any intermediary, as shown in Figure 1 a). However, if the cloud platform doesn't offer a ready-made protocol, content format and information model compatible with the device, it has to be adapted manually (custom API, driver, SDK). Security measures,

https://doi.org/10.34749/3061-1008.2025.2

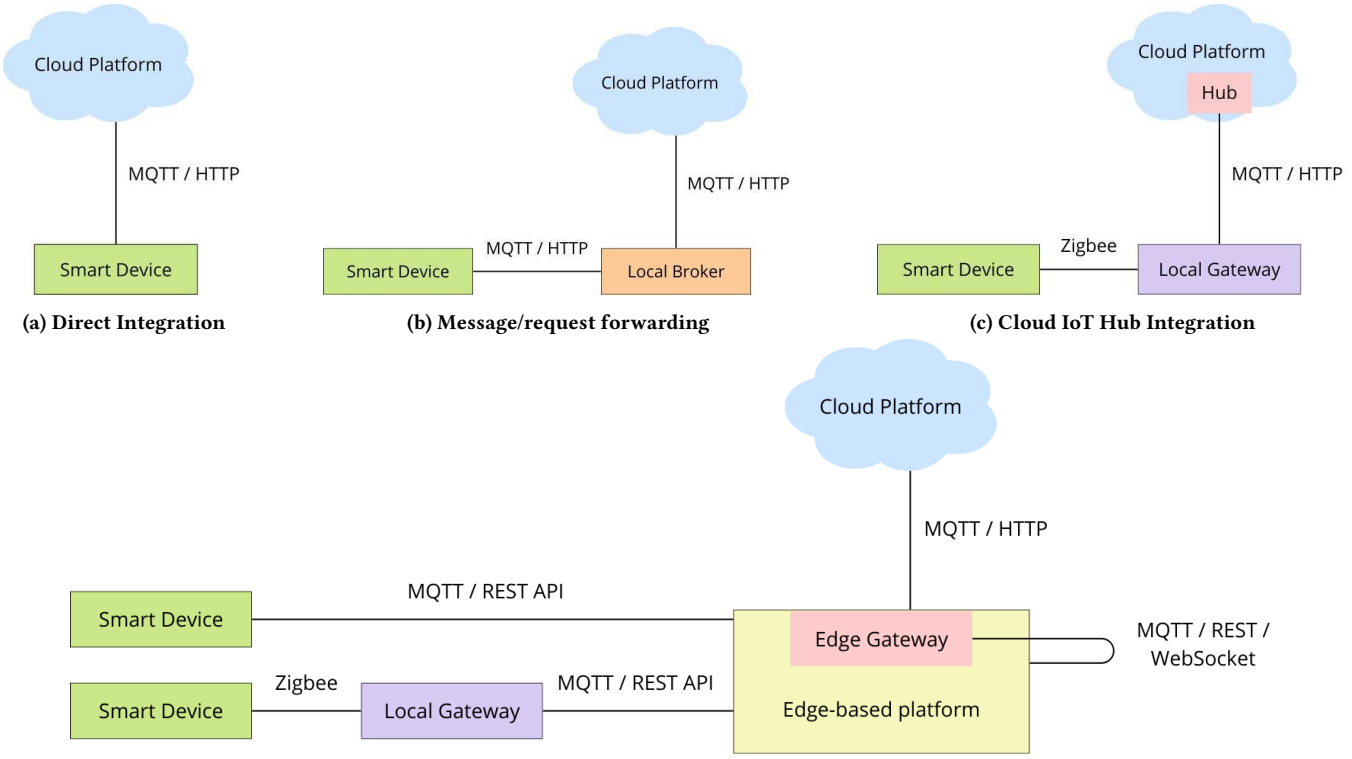


Figure 1: Connecting devices to the cloud-based platform

including TLS certificates, authentication and disconnection handling, must be implemented on the device. In case of no Internet service, the device is left with no local control. This approach is convenient when the device has enough resources to perform the entire integration. However, this approach lacks flexibility for local actuation, as it fully relies on the cloud for rule-based instructions, thereby introducing inherent latency. A practical example of this model can be found in smart assistants such as Amazon Alexa, which connects directly to the AWS platform in the cloud.

3.2 Message/request forwarding

The idea is to have some component in local/edge environment that receives data from IoT devices in local environment and forwards them to cloud platform. One solution is MQTT-forwarding where the local broker acts as an intermediary that forwards messages from end device to a connected cloud MQTT broker, as shown in Figure 1 b). This model is lightweight and relatively simple to deploy, but it has inherent limitations. Message transformation or protocol adaptation is not provided, so devices must produce payloads in a format already compatible with the cloud subscriber. Furthermore, system operation is fully dependent on continuous network connectivity, and this approach is generally restricted to MQTT-capable devices, excluding HTTP or other protocol-based devices. Common implementations of local broker include Mosquitto, which provides reliable message queuing and forwarding for such architectures. Another solution is to use Eclipse Kura in local edge environment [15]. Eclipse Kura is used for protocol translation, and secure connectivity between field devices and cloud platforms. The drawback is that it is service that could be run as part of edge gateway and it is not ready server to install.

3.3 Cloud IoT Hub integration

Compared to direct device-to-cloud integration, where the device itself must implement the protocol expected by the cloud, the IoT hub provides format and protocol translation in the cloud, thus simplifying interoperability. The hub in the cloud acts as a central entry point for all connected devices, as shown in Figure 1 c). Here, the local gateway serves as a protocol translator for devices that use communication protocols without native Internet support. This approach does not support offline operation, since all communication and rule execution depend on the cloud. A major advantage is that the IoT hub can also serve as a bridge for cloud-to-cloud integrations, thereby enabling interoperability with vendor-specific ecosystems (e.g., Sonoff via eWeLink Cloud) and extending device support beyond the native protocols, adding value to the growth of the SISS [12, 13].

3.4 Intermediary Edge Gateway at the Edge-based Platform

As shown in Figure 1 d), data reading from the end device is first sent to the local gateway if the device doesn't operate on same

https://doi.org/10.34749/3061-1008.2025.2

protocols as an edge-based platform and the translation is needed (e.g. connecting IKEA Smart bulbs via IKEA Dirigera hub). Otherwise, device connects directly to the platform. An edge-based platform manages connected devices, applies local control logic and performs filtering or data transformation when necessary. An additional layer typically implemented within the edge-based platform, the edge gateway, acts as a translator and forwarder to the cloud platform, driving the progress of the SISS [12, 13]. It bridges local communication with the protocols and formats expected by the cloud (e.g., JSON over HTTP, MQTT, AMQP), while also handling all security requirements such as TLS certificates, keys or tokens toward cloud platform. Another key feature of the edge gateway is its ability to provide offline buffering, ensuring that any data captured during connectivity loss is stored locally and forwarded to the cloud once the connection is restored. In the case of changing cloud IoT platform just translation layer in edge gateway should be changed. In the case of changing smart device only integration of that device in local plaform need to be changed. This way we have flexible and longstanding deployment. An example of such integration in industrial and more professional scenarios is AWS IoT Greengrass [27] edge computing platform, that has more advanced actions executing at the edge, such as running ML models, but however AWS Cloud provides centralized device management, data synchronization and storage, secure authentication, advanced analytics and model distribution and updates. Similar to that, Microsoft Azure IoT Edge [20] shares this concept, where edge runtime is a local software managing edge modules and communication with the cloud. An open-source solution, ThingsBoard IoT Gateway [32], acts as a bridge between end devices and ThingsBoard platform in the cloud, enabling connection for different types of devices and protocols.

In this paper, we present a case study on integrating Home Assistant with Node-RED as an edge gateway, providing connectivity to the cloud IoT platform Data JediX from Ericsson Nikola Tesla [31]. This setup leverages Home Assistant for local device management and automation, while Node-RED ensures communication with the cloud, making the implementation particularly suitable for home automation scenarios. This architecture revolves around several key aspects: ensuring security by protecting data in transit and authenticating components, maintaining privacy by keeping sensitive information local whenever possible, guaranteeing reliability through mechanisms that prevent data loss and handle connectivity issues, and providing flexibility to handle various data formats and integrate heterogeneous systems. Security requires that the edge component supports TLS/SSL encryption and authentication (e.g., certificates, API keys) with secure protocols such as MQTT over TLS or REST with tokens. Also, gateways have their own credentials. Privacy is addressed through a hybrid edge–cloud approach that keeps sensitive data local (e.g., raw images or video) while sending only aggregated or just necessary information to the cloud, with strict role-based access control in Data JediX. Reliability requires mechanisms to mitigate data loss or delayed real-time processing due to connectivity issues; this is partly ensured by store-and-forward at the edge gateway, MQTT persistent sessions, timestamped data for time-series synchronization and redundancy such as dual connectivity (primary Internet with 4G/5G fallback). Finally, flexibility is achieved by handling heterogeneous data formats (JSON, XML, CSV, binary protocols) at the edge, where data can be translated and harmonized; for instance, mapping a Home Assistant entity into a REST payload consumable by platforms like Data JediX. JediX payload is a syntactic data exchange format which doesn't have predefined semantics.

4 Case study: Home Assistant integration with Data Jedi X via Edge Gateway

Home Assistant is an open source IoT platform with the primary application of smart home automation. It allows connecting and controlling different devices in one local network and runs locally on devices such as Raspberry Pi, thus does not depend on cloud computing services, which enables work without an Internet connection and greater control over data. It supports more than 1,000 integrations, including devices and protocols such as MQTT, Zigbee, Z-Wave, Philips Hue, IKEA TRÅDFRI, Google Assistant and Amazon Alexa. The interface is accessible through a web browser or a mobile application and allows viewing the state of the device, managing and creating automations.

Data JediX is an IoT data management platform, developed entirely within Ericsson Nikola Tesla (ENT) for a broad aspect of applications, such as smart city, industry environments, public health and agriculture. It enables collection, transformation, routing and storage of data from any IoT device, as well as sending messages to devices, applications or users. In addition, the system ensures the protection of data and devices from unauthorized access through advanced security mechanisms. Main functionalities include modular data flows, adapter framework for the transformation of input and output data, an information model for data enrichment and the possibility of advanced analytics and building business rules through the Rule Engine. The platform supports work in virtualized environments, containers, and on physical infrastructure. Key functional blocks include:

(1) **Registry** - for hierarchical management of devices, users and applications
(2) **Data Repository** - for secure data storage, independent of the protocol
(3) **Subscription/Notification system** - for managing subscriptions and notifications in real time
(4) **Workflow Engine and Data Transformation module** - for adjusting the flow and data format in real time
(5) **Rule Engine** - for defining automated business rules
(6) **Analytics module** - for basic and advanced data processing
(7) **Intuitive user interface** - for entity introduction, configuration and visualization of data
(8) **Security controls** - including authentication, authorization and login

To register devices on the Data JediX platform, there is a need to establish the organizational structure, which begins with defining the operator (e.g., `SmartEnvironment`), the associated domain (e.g., `SmartUniversity`), the organization (e.g., `FER_org`) and the user entity (e.g., `FER_Departments`). Within this framework, a gateway group (`FER_building_A`) is created to represent the physical

https://doi.org/10.34749/3061-1008.2025.2

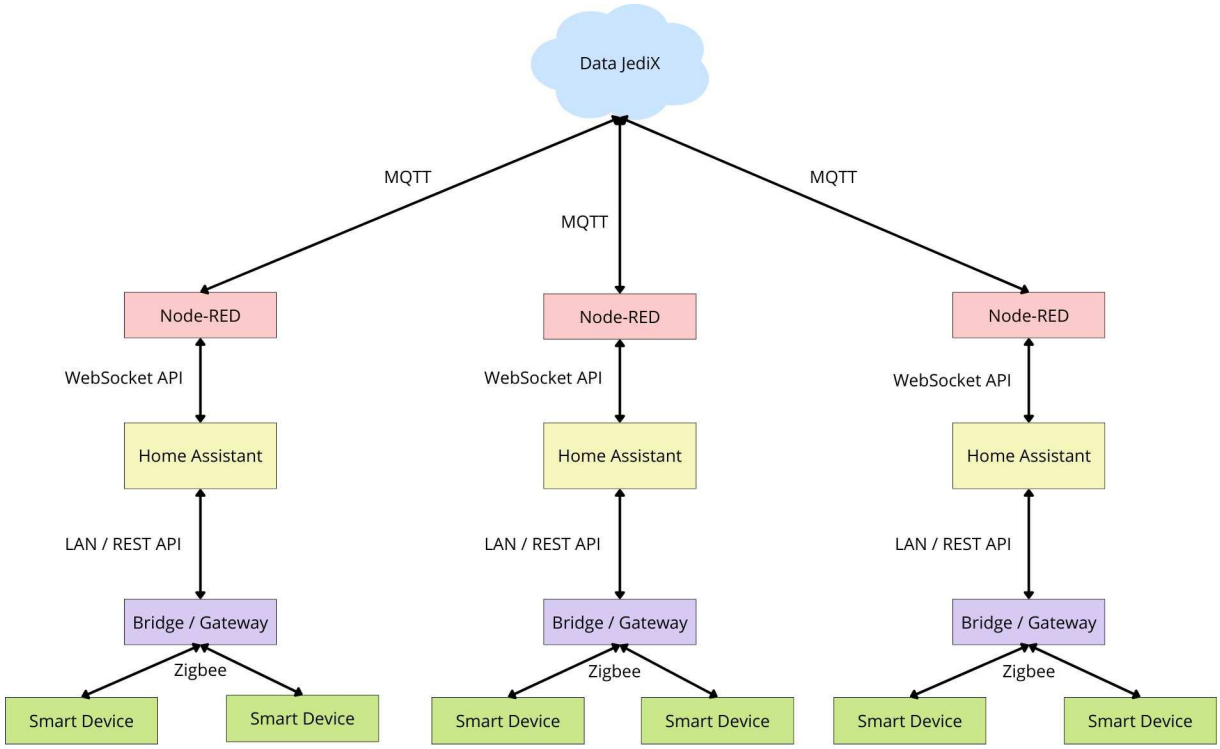


Figure 2: Architecture of the system integrating local and in-the-cloud platform through edge gateway

location in which the devices are deployed. Once the organizational layer is defined, specifications are created to serve as templates for different types of components: the gateway group specification (e.g., `UniversityBuilding_GW_GRP`), the gateway specification (e.g., `HomeAssistant_GW_SPEC`), sensor specifications (e.g., `HueSmartLight_Spec`, `HueMotionSensor_Spec`), and corresponding resource specifications (e.g., `HueSmartLightRes_Spec`, `HueMotionSensorRes_Spec`). Based on these templates, concrete instances are then added. Two gateways are registered (e.g., `Home_Assistant_1` and `Home_Assistant_2`), each linked to the `FER_building_A` group and the gateway specification. Individual sensors, such as `HueGoLamp_1`, `HueColorLamp_1`, and `HueMotionSensor_1` for the first gateway, or `HueGoLamp_2`, `HueColorLamp_2`, and `HueTemperature` for the second, are associated with the appropriate gateway and sensor specification. Finally, resources are defined to capture the data of each sensor (e.g., `HueMotionSensor_1_Res` for the motion sensor `HueMotionSensor_1`), each connected to its corresponding resource specification. With this hierarchical structure in place, devices become fully registered on the platform and ready for subsequent automation through subscriptions and rules.
Node-RED is an open-source, JavaScript-based flow programming environment, originally developed within IBM and now part of the OpenJS Foundation. It enables easy connection of devices, services and applications through a visual interface. This makes it suitable for applications ranging from smart home automation to industrial control systems. Its low-code nature allows users of varying technical expertise to create system logic by connecting visual nodes representing inputs, conditions, data processing and outputs. Node-RED runs locally on edge devices like Raspberry Pi, on cloud platforms such as AWS or Azure and via containers, leveraging Node.js for efficiency. It supports standard protocols including MQTT, HTTP, WebSocket and TCP, stores flows in JSON format for easy sharing, and offers an online library with ready-made integrations, including Home Assistant, cloud platforms and databases [16].
The architecture of the realized integration is shown in Figure 2. At the local layer, Home Assistant manages Philips Hue devices via the Zigbee Hue Bridge and enables local automation and two-way communication with devices without the need for the Internet. Node-RED utilizes WebSocket API for communicating with the Home Assistant, while it connects Home Assistant and Data JediX using MQTT. A remote MQTT broker on the Data JediX platform enables communication between the local and cloud layers, with Node-RED acting as an MQTT client, ensuring reliable, asynchronous two-way data exchange. Through this common broker, Data JediX centrally registers local instances of Home Assistant, offering unified device management, automation, data storage, analytical reporting and system control across multiple locations.

The functionality of the solution is shown in Figure 3, where a sequence diagram shows in detail the communication flows between system components. The Data JediX platform tracks values coming from local motion sensor. When a true value is detected on one of the sensors, a command is sent to turn on all smart lamps on all connected Home Assistant instances. The command is sent via MQTT to a topic subscribed to by all local systems. Each local

https://doi.org/10.34749/3061-1008.2025.2

system receives the message, recognizes the command and turns on the local smart lamps. To achieve this functionality, a flow was created in the Node-RED. The first part of the flow is about retrieving the state of the motion sensor from the Home Assistant platform. In the initial part of the flow, a node of type `inject` is used, for the purpose of initiating the execution of the data flow at regular time intervals. In this system, the `inject` node is configured to generate a new message every 5 seconds that activates the retrieval of the current state of the motion sensor. The node is configured so that the repeat time option is selected in the Repeat field, with the value set to 5 seconds. The payload and topic fields are left empty because they are not needed for this operation. This node is connected directly to the `api-current-state` node which retrieves the state of the motion sensor `binary_sensor.hue_motion_sensor_1_motion` from the Home Assistant platform. Then a node called `api-current-state` belonging to the group of nodes that come with the `node-red-contrib-home-assistant-websocket` plugin is used, which allows direct integration with the Home Assistant platform via its WebSocket API connection. The `api-current-state` node allows retrieving the current state of any entity in Home Assistant. In the configuration of the node, it is necessary to specify the entity that we want to read. In this case, the motion sensor identifier `binary_sensor.hue_motion_sensor_1_motion` is entered. After this node receives the data from the Home Assistant platform, the message is sent to two nodes. The first of these is the `debug` node, which is used to print data in the sidebar of the Node-RED interface to check the accuracy of the flow execution. The second node is a function node named "DataJediX format". A `Function` node is a general node type in Node-RED that allows logic to be written in JavaScript. In this case, it is used to transform the data format from Home Assistant to the format defined by the Data JediX platform standards. Inside the node, a JSON object is formed with a special structure consisting of an array of `contentNodes`. Each element of that array represents a single piece of data that is sent to the platform. The key part of each element is the `source` object, which must contain the `resource` property because it identifies exactly which resource is sending the data, as shown in Listing 1.

Listing 1: JSON object content inside node Function

```
msg.payload = {
    contentNodes: [
        {
            source: {
                resource: "MotionSensorRes"
            },
            value: msg.payload === "on" ? "true" :
                 "false"
        }
    ]
};
return msg;
```

Without this identifier, the Data JediX platform cannot properly associate the received data with the corresponding device or sensor. In addition to the `source` object, each element also contains a `value` property that carries the actual value of the data. In the specific case of the motion sensor, the value is set to `true` if the sensor is activated, or `false` for all other cases. This transformation ensures that the data matches the expected format and data type that the platform expects. The output from this `Function` node is connected to the `mqtt out` node, which sends a prepared message to the remote MQTT broker. The `mqtt` node is used to publish messages on a specific MQTT topic. In this case, the `digiphy1/in/HueMotionSensor_1` topic is used. Topic indicates the path of data on the Data JediX platform, where `digiphy1` represents the user, `in` is the input channel for data and `HueMotionSensor_1` is the specific name of the sensor. When configuring the `mqtt out` node, it is necessary to define the MQTT broker beforehand. In this case, a remote broker at the address `djx.entlab.hr` and `port 1883` is used. After the node is configured, it is possible to publish messages on topics that we define ourselves. In this way, the actual readings from the motion sensor are sent from Home Assistant to the Data JediX platform in the prescribed format. In addition to sending data to the cloud, the system also enables receiving management commands from the cloud, which come from the Data JediX platform via MQTT. For this, the `mqtt in` node is used, which listens to the `digiphy1/out/datajedix_incoming` topic, where `digiphy1` denotes the user, `out` represents the output path for data from the Data JediX platform, and `datajedix_incoming` is the name of the topic that we arbitrarily chose on the Data JediX platform as the destination for data from the subscription. Messages arriving at this node are simultaneously forwarded to two outputs. The first goes to the `debug` node that monitors incoming data, while the second leads to the `Function` node that checks whether the message contains a command to activate the light on smart lamps, as shown in Listing 2.

Listing 2: Function to check for the presence of a command to activate the light

```
if (
  msg.payload &&
  Array.isArray(msg.payload.contentNodes)
) {
  const effectNode = msg.payload.contentNodes.find(node
       =>
    node.metadata &&
    node.metadata.name === "TurnOnLampEffect" &&
    node.value === true
  );

  if (effectNode) {
    return msg;
  }
}
return null;
```

A simple check is implemented in the `Function` node. If the message contains an array of `contentNodes`, the function searches within that array for an element that has metadata named `TurnOnLampEffect` with the value true. If such an element exists, the message is forwarded, otherwise no action is taken. If the condition in the `Function` node is met, the message is sent to the `api-call-service` node that calls the Home Assistant service `light.toggle`. In the configuration, it is defined that the service `light.toggle` applies to two entities: `light.hue_color_lamp_1` and `light.hue_go_1`. This makes it possible to turn on the `HueColorLamp_1` and `HueGoLamp_1` smart lamps. When configuring this node, it is necessary to enter the name of the *light.toggle* service in the `Service` field and enter the identifiers of all entities in the `Entity ID` field. We do the same when configuring the action node in the second

https://doi.org/10.34749/3061-1008.2025.2

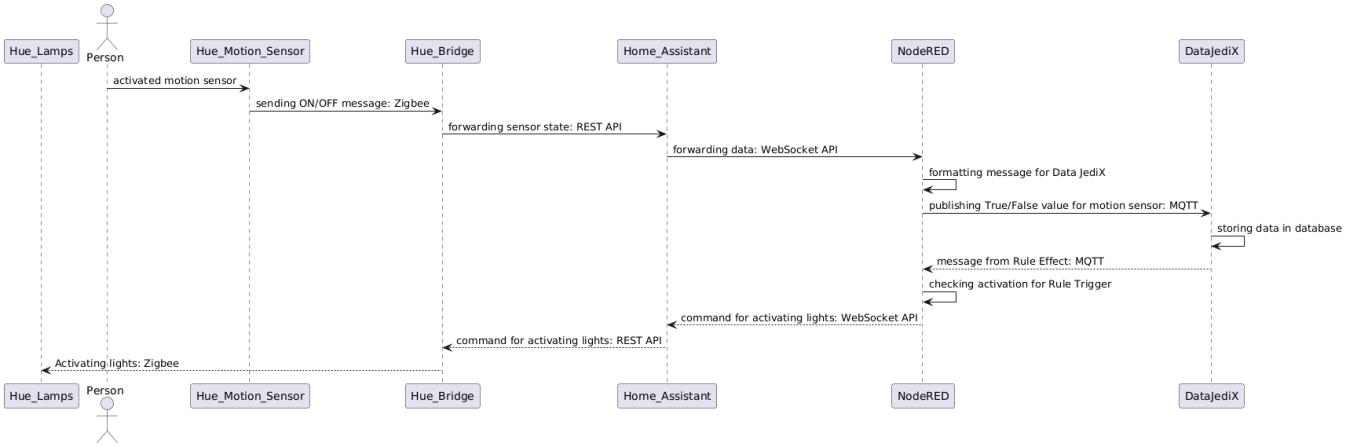


Figure 3: Automation sequence diagram from the perspective of the local environment in which the motion sensor activation occurred

flow for the second instance of Home Assistant. The second configuration allows us to turn on smart lamps that are connected to another instance of Home Assistant. These are the smart lamps `HueGoLamp_2` and `HueColorLamp_2`.

4.1 Local data processing

Before applying format transformation to the incoming data, Node-RED is capable of filtering based on predefined criteria. In the presented case study, "Collect & filter data" `Function` node accesses previously stored device states from the flow context. Incoming values from `msg.payload`, received as strings, are parsed to the float data type. The flow then checks if the value is a valid number. If it is, the value is appended to the stored data array and updated in the flow context. This ensures that only numerically valid sensor data is collected, providing a clean and consistent dataset for subsequent processing. Other than filtering, additional operations like aggregation or computing minimum/maximum values can be performed on the collected data. For example, the "Avg value" `Function` node computes the average temperature every fifteen minutes. It first retrieves the recorded sensor readings from the flow context into a local array, filtering out any non-numeric values. If the array is empty after filtering, the node returns null and no message is forwarded. The remaining numeric values are summed and divided by the array length to calculate the average. This value is then rounded to one decimal place, assigned to `msg.payload` and sent to the next node. After this, the stored values in the flow context are cleared to prepare for the next cycle, as shown in Listing 3.

Listing 3: Filtering the data and calculating the mean value

```
let data = flow.get("values") || [];

data = data.filter(v => typeof v === 'number' &&
     !isNaN(v));

if (data.length === 0) {
  return null;
}

let sum = data.reduce((a, b) => a + b, 0);
let avg = sum / data.length;
avg = parseFloat(avg.toFixed(1));
msg.payload = avg;
flow.set("values", []);

return msg;
```

The application of filtering and data processing operations can significantly reduce network traffic and enhance privacy, as some data remain confined to the edge-based platform. Moreover, it mitigates the risk of potential adversaries capturing raw data, since only aggregated or otherwise processed values are transmitted. This hybrid solution also supports automated actuation, which, when executed locally by Home Assistant, reduces latency and becomes an integral part of the local processing capabilities.

4.2 Interoperability

Interoperability can be understood as the ability of diverse systems, devices and organizations to communicate, exchange data and use it in a consistent and meaningful way. At the technical level, this requires alignment of hardware and software through shared protocols, interfaces and data formats to enable effective information exchange. Syntactic interoperability addresses the structure of data, typically by relying on standardized representations such as XML or JSON, while semantic interoperability ensures that exchanged information carries the same meaning for all parties by adopting common data models or ontologies. In addition, organizational and legal interoperability refer to the policies, agreements and regulatory frameworks that make collaboration across different entities possible. In the IoT domain, interoperability is particularly critical due to the heterogeneity of devices and platforms, which often operate on dissimilar protocols or data formats. Introducing an edge gateway between local systems and the cloud helps bridge these differences, ensuring both technical and syntactic interoperability. In the presented case study, communication between the Home Assistant system and the Data JediX platform relies on the MQTT protocol at the transport layer, where messages are published and received on predefined topics (e.g. `digiphy1/in/...` for

https://doi.org/10.34749/3061-1008.2025.2

input data and `digiphy1/out/...` for output data). The content of each MQTT message must be formatted in JSON format that is compliant with the standards of the Data JediX platform. The key component of this format is the `contentNodes` field, which represents an array of elements, where each element contains an object `source` with the `resource` property that uniquely identifies the data source and the `value` field that carries the actual value of the sensor reading. Additionally, elements can contain a `metadata` field, which is used to transmit control commands and effects (e.g. `TurnOnLampEffect`). In this way, data from the sensor is transformed into syntax of the prescribed JSON format using the `Function` node in Node-RED and sent to the Data JediX platform, while commands from the platform come in the same format and are forwarded to Home Assistant services for execution.

5 Conclusion

Analysis of existing connectivity strategies indicates that a two-tier architecture with an Intermediary Edge Gateway provides an effective balance between local autonomy and global scalability. This approach enables cloud-adapted payload transformation, while enhancing data throughput efficiency with filtering and aggregation. A case study involving the open-source edge platform Home Assistant, the commercial Data JediX cloud platform, and Node-RED as an intermediary demonstrates that the proposed architecture enables scalable, modular management of heterogeneous IoT devices. A key advantage of the described IoT integration lies in its durability: the edge (local) platform can remain fixed, while the cloud provider can be changed or updated over time. This approach ensures stability at the edge for connected devices while providing flexibility to switch or upgrade cloud services as needed. By combining local preprocessing with centralized processing, this hybrid solution enhances interoperability, reduces reliance on network connectivity, and improves operational efficiency, providing a solid foundation for persistent IoT solutions. However, the presented case study serves primarily as an initial proof of concept. Building upon this foundation, future work could include a comprehensive empirical evaluation encompassing key performance metrics such as latency, throughput, scalability, expected network traffic reduction, and CPU utilization on the gateway. Furthermore, extending the system to integrate with other cloud-based IoT platforms would enable assessment of its generalizability and interoperability across diverse deployment environments.

6 Acknowledgments

This research has been funded by the European Union – NextGenerationEU, under the project XR Communication and Interaction Through a Dynamically Updated Digital Twin of a Smart Space – DIGIPHY, grant number NPOO.C3.2.R3-I1.04.0070. Views and opinions expressed are however those of the authors only and do not necessarily reflect those of the European Union or the European Commission. Neither the EU nor the EC can be held responsible for them.

References

[1] A.A.H. Al-Shateri et al. 2024. Luna: A Benchmark Project in the Convergence of Artificial Intelligence and Internet of Things for Home Automation. In *2024 IEEE International Conference on Advanced Telecommunication and Networking Technologies (ATNT)*, Vol. 1. 1–4. doi:10.1109/ATNT61688.2024.10719226

[2] F.C. Andriulo et al. 2024. Edge Computing and Cloud Computing for Internet of Things: A Review. *Informatics* 11, 4 (2024). https://www.mdpi.com/2227-9709/11/4/71

[3] N.A. Angel et al. 2022. Recent Advances in Evolving Computing Paradigms: Cloud, Edge, and Fog Technologies. *Sensors* 22, 1 (2022). https://www.mdpi.com/1424-8220/22/1/196

[4] Home Assistant. 2025. Home Assistant Integrations - Amazon Web Services (AWS). https://www.home-assistant.io/integrations/aws/. Accessed: 2025-09-09.

[5] Home Assistant. 2025. Home Assistant Integrations - Azure Event Hub. https://www.home-assistant.io/integrations/azure_event_hub/. Accessed: 2025-09-09.

[6] Home Assistant. 2025. Home Assistant Integrations - Documentation. https://www.home-assistant.io/docs/. Accessed: 2025-09-09.

[7] Home Assistant. 2025. Home Assistant Integrations - Google Cloud. https://www.home-assistant.io/integrations/google_cloud/. Accessed: 2025-09-09.

[8] Home Assistant. 2025. Home Assistant Integrations - IBM Watson IoT Platform. https://www.home-assistant.io/integrations/watson_iot/. Accessed: 2025-09-09.

[9] M. Bauer. 2022. FIWARE: Standard-based Open Source Components for Cross-Domain IoT Platforms. In *2022 IEEE 8th World Forum on Internet of Things (WF-IoT)*. 1–6. doi:10.1109/WF-IoT54382.2022.10152259

[10] F. Cirillo et al. 2019. A Standard-Based Open Source IoT Platform: FIWARE. *IEEE Internet of Things Magazine* 2, 3 (2019), 12–18. doi:10.1109/IOTM.0001.1800022

[11] N. Das and M. Shakil. 2025. Design and Implementation of an IoT based Energy-Efficient User-Friendly Smart Home Automation System for Energy Savings and Maximizing Comfort. 1–5. doi:10.1109/ECCE64574.2025.11013781

[12] M.S.E. de la Parte et al. 2023. Breaking Down IoT Silos: Semantic Interoperability Support System for the Internet of Things. In *2023 3rd International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME)*. 1–7. doi:10.1109/ICECCME57830.2023.10253024

[13] M.S.E. de la Parte et al. 2025. SISS: Semantic Interoperability Support System for the Internet of Things. *IEEE Internet of Things Journal* 12, 16, 33769–33791. doi:10.1109/JIOT.2025.3577776

[14] O. Debauche et al. 2022. A New Edge Computing Architecture for IoT and Multimedia Data Management. *Information* 13, 2 (2022). doi:10.3390/info13020089

[15] Eclipse. 2025. Eclipse Kura project. https://projects.eclipse.org/projects/iot.kura. Accessed: 2025-09-15.

[16] Nijhof F. 2025. Home Assistant Community Add-on: Node-RED. https://community.home-assistant.io/t/home-assistant-community-add-on-node-red/55023/1. Accessed: 2025-09-09.

[17] K. Habib et al. 2022. An Aggregated Data Integration Approach to the Web and Cloud Platforms through a Modular REST-Based OPC UA Middleware. *Sensors (Basel)* 22, 5 (2022).

[18] G. Hatzivasilis et al. 2019. Secure Semantic Interoperability for IoT Applications with Linked Data. In *2019 IEEE Global Communications Conference (GLOBECOM)*. 1–6. doi:10.1109/GLOBECOM38437.2019.9013147

[19] P. Schoutsen Home Asisstant. 2025. Classifying the Internet of Things. https://www.home-assistant.io/blog/2016/02/12/classifying-the-internet-of-things/#classifiers. Accessed: 2025-09-09.

[20] Microsoft. 2025. What is Azure IoT Edge. https://learn.microsoft.com/en-us/azure/iot-edge/about-iot-edge. Accessed: 2025-09-09.

[21] openHAB. 2025. Add-on Reference, Bindings. https://www.openhab.org/addons/. Accessed: 2025-09-09.

[22] openHAB. 2025. openHAB - Documentation. https://www.openhab.org/docs/. Accessed: 2025-09-09.

[23] V.N. Pham et al. 2021. Efficient Solution for Large-Scale IoT Applications with Proactive Edge-Cloud Publish/Subscribe Brokers Clustering. *Sensors* 21, 24 (2021).

[24] I. Podnar Žarko et al. 2017. Towards an IoT framework for Semantic and Organizational Interoperability. In *2017 Global Internet of Things Summit (GIoTS)*. 1–6. doi:10.1109/GIOTS.2017.8016253

[25] I. Podnar Žarko et al. 2019. The symbIoTe Solution for Semantic and Syntactic Interoperability of Cloud-based IoT Platforms. In *2019 Global IoT Summit (GIoTS)*. 1–6. doi:10.1109/GIOTS.2019.8766420

[26] P. Savaridass et al. 2025. A Multi-Protocol IoT-Enabled Touch Control Switch for Energy-Efficient Smart Home Automation. In *2025 Second International Conference on Cognitive Robotics and Intelligent Systems (ICC - ROBINS)*. 133–139. doi:10.1109/ICC-ROBINS64345.2025.11086234

[27] Amazon Web Services. 2025. AWS IoT Greengrass, Developer Guide, Version 2. https://docs.aws.amazon.com/pdfs/greengrass/v2/developerguide/greengrass-v2-developer-guide.pdf#what-is-iot-greengrass. Accessed: 2025-09-09.

[28] J. Shiraishi et al. 2024. Coexistence of Push Wireless Access with Pull Communication for Content-based Wake-up Radios. In *GLOBECOM 2024 - 2024 IEEE Global Communications Conference*. 4836–4841. doi:10.1109/GLOBECOM52923.2024.10901717

[29] I. V. Sita and B. David. 2024. Development and Implementation of a Comprehensive Smart Home Automation System Using Home Assistant and IoT Devices. In

https://doi.org/10.34749/3061-1008.2025.2

2024 4th International Conference on Electrical, Computer, Communications and Mechatronics Engineering (ICECCME). 1–7. doi:10.1109/ICECCME62383.2024.10796669

[30] E. O. Sodiya et al. 2024. Current State and Prospects of Edge Computing within the Internet of Things (IoT) Ecosystem. *International Journal of Scientific Research and Analysis* 11, 1 (2024), 1863–1873. https://ijsra.net/sites/default/files/IJSRA-2024-0287.pdf

[31] Ericsson Nikola Tesla. 2025. Data Management Platform. https://ericssonnikolatesla.com/en/digital-society/platforms/data-management-platform/. Accessed: 2025-09-09.

[32] ThingsBoard. 2025. What is ThingsBoard IoT Gateway? https://thingsboard.io/docs/iot-gateway/what-is-iot-gateway/. Accessed: 2025-09-09.

[33] A. Tolcha et al. 2021. Towards Interoperability of Entity-Based and Event-Based IoT Platforms: The Case of NGSI and EPCIS Standards. *IEEE Access* 9, 49868–49880. doi:10.1109/ACCESS.2021.3069194

  https://doi.org/10.34749/3061-1008.2025.2