Ref. Ares(2023)1659700 - 07/03/2023

H2020 – LC – SC3 – EE – 2019 – GA 894240

Operating System for Smart Services in Buildings



D2.6 Upgraded cloud.iO Platform

WP2 IoT for Smart Buildings

	Name	Date
Prepared by	Dominique Gabioud (HES-SO), Martin Meyer (HES-SO), Frédéric Revaz (HES-SO), Pascal Chaussumier (EDF), Luc Dufour (EDF)	24.01.2023
Peer reviewed by	Till Richter (ALIUNID), Matija Arh (INEA)	02.02.2023
Reviewed and approved by	Dominique Gabioud (HES-SO)	06.03.2023



This project has received funding from the European Union's Horizon 2020 research and innovation program under grant agreement No 894240.

Distribution list

External		Internal	
European Commission	x	Consortium partners	x

Change log

Version	Date	Remark / changes
1.1	24.01.2023	First version by HES-SO & EDF
2.1	11.02.2023	Second version by HES-SO & EDF
3.1	06.03.2023	Final version by HES-SO

To be cited as

"D2.6 Upgraded cloud.iO Platform" of the HORIZON 2020 project domOS, EC Grant Agreement No 894240.

Disclaimer

This document's contents are not intended to replace the consultation of any applicable legal sources or the necessary advice of a legal expert, where appropriate. All information in this document is provided "as is" and no guarantee or warranty is given that the information is fit for any particular purpose. The user, therefore, uses the information at its sole risk and liability. For the avoidance of all doubts, the European Commission has no liability in respect of this document, which is merely representing the authors' view.

Table of Contents

Executive Summary **6**

1. Introduction **6**
1.1. From the domOS IoT Architecture to the Upgraded Platforms 6
1.2. IoT Platforms and Demonstrators 6
1.3. Content of the Document 7

2. Implementation of cloud.iO Platform **7**
2.1. Overview of the Platform 7
2.1.1. Collection and Storage of Building Metadata. 10
2.2. Description of Platform Implementation 14
2.2.1. Generation of BDs 14
2.2.2. Generation of TDs 14
2.2.3. Implementation 15
2.2.4. Reachability of the BDs 15
2.2.5. Reachability of the TDs 16
2.2.6. Description of Platform Security and Privacy Mechanisms 16
2.2.7. Description of Platform Management 17

3. Implementation of the EDF Platform **18**
3.1. Overview of the Platform 18
3.1.1. Architecture 18
3.1.2. Implementation 20
3.1.3. Zigbee 21
3.1.4. MQTT Topics and Data Model 21
3.2. Description of Platform Implementation 22
3.2.1. Reachability of the BDs 22
3.2.2. Reachability of TDs 22
3.2.3. Generation of the TDs and BDs 22
3.2.4. Description of Platform Security and Privacy Mechanisms 23
3.2.5. Description of Platform Management 24

4. Conclusion **24**

5. References **24**

List of figures

Figure 1: Components in a cloud.iO System 8
Figure 2: Architecture of the cloud.iO Platform....8
Figure 3: Object Oriented Data Model of cloud.iO Endpoints 9
Figure 4: UML Class Diagram for the cloud.iO Class Attribute....9
Figure 5: Concepts of the Database for Metadata.... 10
Figure 6: Structure of the `Url` Table.... 11
Figure 7: Principle for Dynamic Properties 12
Figure 8: Schema of the Relational Database for the Database for Metadata 13
Figure 9: Process to Generate BDs.... 14
Figure 10: TD Generation Process.... 15
Figure 11: Application Deployment in the EDF System.... 19
Figure 12: Architecture of the EDF Platform with Local Infrastructure 19
Figure 13: Mobile Application Display on the Play Store.... 20
Figure 14: Architecture of The EDF IoT Platform 21
Figure 15: Format for MQTT Topics 22
Figure 16: Sample JSON Encoded MQTT Payload 22

List of tables

Table 1: Changes in IoT Platforms for Demonstrators and Their Motivation 6
Table 2: References for the Open-Source cloud.iO Platform....7
Table 3: Main Features of the cloud.iO IoT Platform....8
Table 4: Features of the cloud.iO Endpoint Library 8
Table 5: Commentary for Concepts of the Metadata Database 11
Table 6: Sample Entry in the `Url` Table 11
Table 7: Main Features of the EDF IoT Platform 20
Table 8: Features of the EDF System Endpoint 21

Terms, definitions, and abbreviated terms

AFE	Available Flexibility Estimator
API	Java Scripting API
BD	Building Description
dCO	domOS Core Ontology
ERL	Emetteur Radio Local
IoT	Internet of Things
GA	Grant Agreement
JSON-LD	JSON for Linking Data
NAT	Network Address Translation
RDF	Resource Description Framework
PAN	Personal Area Network
PAT	Port Address Translation
SHACL	Shapes Constraint Language
SPARQL	SPARQL Protocol And RDF Query Language
TD	Thing Description
TM	Thing Model
WoT	Web of Thing
W3C	World Wide Web Consortium
ZCL	Zigbee Cluster Library

Executive Summary

This report documents how the Sion and Paris demonstrators implement the software components to comply with the domOS IoT ecosystem functional specification. During the definition part of the project, it was initially envisioned that HES-SO would provide EDF with an upgraded platform for the Paris demonstrator. However, due to internal policies and new business developments, it was decided that EDF would implement its own domOS compliant platform during the project. This document reflects the evolution of the project and presents two platforms in one document. The purpose of the present report is to document the technical developments that have been performed to upgrade two platforms: the cloud.io platform and the EDF platform.

The present document reports the major design choices that were made when upgrading the two platforms, such as type of platform (“open” or “closed”), and how the domOS compliance is achieved.

In conclusion, the two platforms are operational, compliant and the core features are implemented.

1. Introduction

1.1. From the domOS IoT Architecture to the Upgraded Platforms

The domOS IoT architecture as defined in the Deliverable D2.3 (Nielsen, et al., 2022) is a specification that allows IoT platforms to decouple the application layer and the in-building infrastructure in smart buildings. To maximise the chance of success of this specification, the domOS team designed an architecture that is compliant with existing and emerging IoT and ICT standards and requires limited effort for the update of existing platforms.

1.2. IoT Platforms and Demonstrators

Several changes external to domOS impact the planned use of upgraded IoT platforms for demonstrators. As they all concern the cloud.iO platform, the changes are reported in the present document.

TABLE 1: CHANGES IN IOT PLATFORMS FOR DEMONSTRATORS AND THEIR MOTIVATION

Demonstration site	Upgraded IoT platform indicated in the DoA	Upgraded IoT platform(s) effectively used	Motivation for change
Paris	cloud.iO	EDF IoT	1. The EDF IoT platform offers services basically similar to cloud.iO. 2. It is not possible to perform the acceptance procedure required by EDF for new IoT platforms with the resources and the timeframe of the project. 3. The porting of the IoT architecture to an additional platform is a valuable result for domOS.
Sion	S-IOT	cloud.iO	The gateways inherited from the Sion demonstrator of the GOFLEX project were

			powered by cloud.iO. Their upgrade to S-IOT would not have brought any added value.
		S-IOT	New ALIUNID gateways operated by S-IoT are installed in addition. To showcase, the independence between the application layers and the field layers, services will be able to operate transparently over cloud.iO and S-IOT gateways, provided the required sensors and actuators are present.
Neuchâtel	cloud.iO	-	The owner of the (unique) building in the Neuchâtel site decided to terminate the demonstration in Q4 2022, at a time were the deployment of cloud.iO was still in its planning phase.

1.3. Content of the Document

The present document describes how the cloud.iO (Chapter 2) and EDF IoT (Chapter 3) platforms have been upgraded to the domOS IoT specification as defined in the Deliverable D2.x "Functional Specification of the IoT-Ecosystem".

2. Implementation of cloud.iO Platform

2.1. Overview of the Platform

cloud.iO is a low-complexity, scalable open-source IoT (Internet of Things) platform developed by HES-SO. The references to the publicly available sources are indicated in Table 2.

TABLE 2: REFERENCES FOR THE OPEN-SOURCE CLOUD.IO PLATFORM

License	MIT
Documentation	https://cloudio.hevs.ch
Repository Git	https://github.com/cloudio-project
Logo	

A cloud.iO IoT system is made up of three components:

1. a central system named "cloud.iO platform",
2. distributed "things" called "cloud.iO endpoints",
3. applications monitoring and/or controlling endpoints.

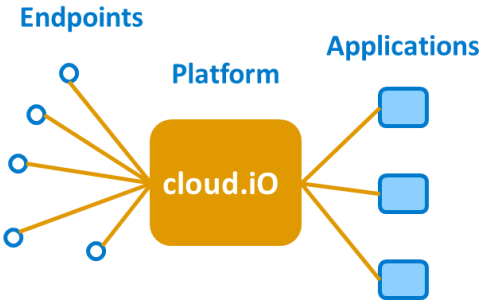


FIGURE 1: COMPONENTS IN A CLOUD.IO SYSTEM

The main features of the cloud.iO platform are summarised in Table 3.

TABLE 3: MAIN FEATURES OF THE CLOUD.IO IOT PLATFORM

Architecture	Micro-service	
Micro-service platform	Spring Boot	
Programming language	Kotlin	
Main components	Relational database management system:	PostgreSQL
	Time series database management system:	InfluxDB
	Message broker	RabbitMQ
Deployment	Docker containers available from Docker Hub	

cloud.iO endpoints must have an IP connectivity to their cloud.iO platform (intermediate Network Address Translation / Port Address Translation (NAT / PAT) supported). Endpoints libraries (see features in Table 4) are available in several programming languages. Developers use them to develop ad hoc solutions tailored to a specific context.

TABLE 4: FEATURES OF THE CLOUD.IO ENDPOINT LIBRARY

Language for endpoint library	Java, Python, C++
Communication library	Paho

The structure of the cloud.iO platform is presented in Figure 2.

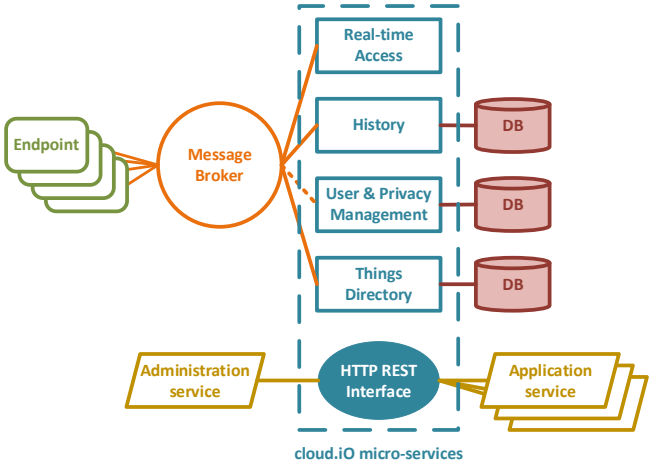


FIGURE 2: ARCHITECTURE OF THE CLOUD.IO PLATFORM

Endpoints are connected to the message broker of the platform using the MQTTS protocol, with certificate-based endpoint authentication. Endpoints features the tree-shaped object-oriented data model represented in Figure 3.

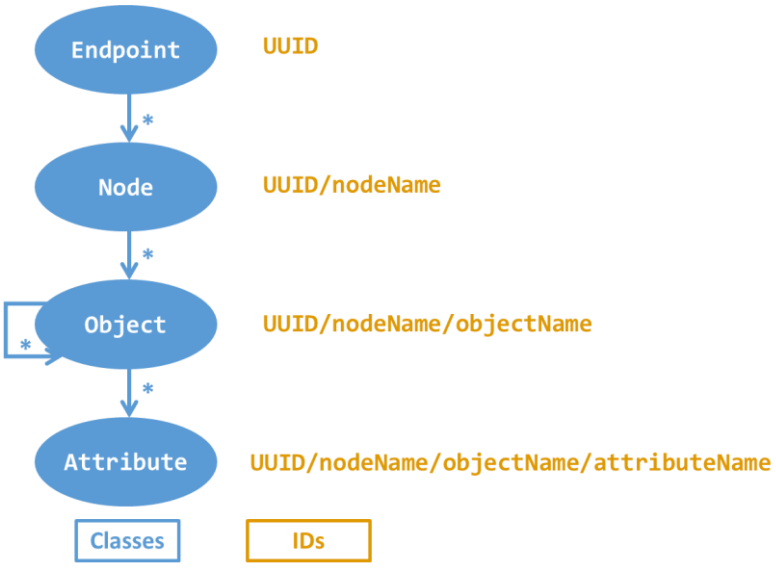


FIGURE 3: OBJECT ORIENTED DATA MODEL OF CLOUD.IO ENDPOINTS

Attributes features a basic type (number, string). When updated by an endpoint, readable attributes, typically measurement results or statuses, are automatically pushed to the platform by the cloud.iO endpoint library. Conversely, the platform can update writable attributes, set points or configuration parameters, at any time, the endpoint library is then responsible for handling the update according to its context.

cloud.iO attribute identifiers – see example in Figure 3 – are part of the MQTTS topic, whereas attribute values are exchanged in MQTTS payload, with JSON or CBOR encoding. cloud.iO attributes are indeed objects (according to the object-oriented pattern) with 4 attributes, as illustrate in the class diagram of Figure 4.

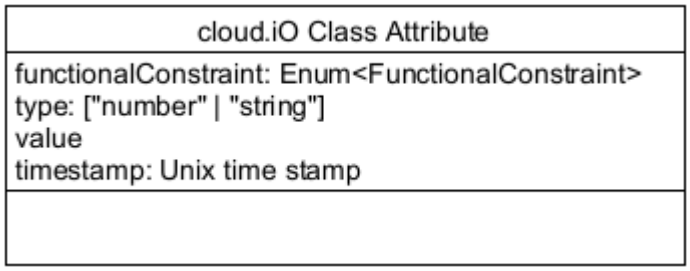


FIGURE 4: UML CLASS DIAGRAM FOR THE CLOUD.IO CLASS ATTRIBUTE

The attributes of the cloud.iO class `Attribute` are serialised for read / write operations in MQTTS payloads using JSON or CBOR.

Endpoints are identified by UUIDs.

Applications can read or write attributes in real-time, subscribe to attribute changes and be notified of changes, as well as access historical values of attributes. In cloud.iO, an access control mechanism enables an administrator to assign read / write access rights down to the attribute level.

Applications can access the platform either through MQTTS or HTTPS.

cloud.iO can – but must not – have an elementary semantics: endpoints can declare objects, which are containers for attributes, as members of a class. They can also declare that a node – an object container – as compliant with an interface (i.e., the node must contain at least all objects specified in the interface).

In domOS, each building features a gateway hosting a cloud.iO endpoint. Each monitoring or control device is a node. Data within a device are structured in objects and in attributes within objects.

2.1.1. Collection and Storage of Building Metadata.

In a domOS building, a gateway hosting cloud.iO endpoints connect a cluster of local appliances and/or devices acting as cloud.iO nodes.

The cloud.iO elementary semantics is too limited to express the metadata needed for BDs. For example, building structure description elements, monitoring devices and the processes they monitor, are required to craft BDs. Hence, additional data must be collected and stored in an additional database. The schema of this database must provide flexibility as the structure of the data to be stored can evolve over time. A MySQL relational database has been deployed for this purpose. Its schema has been designed to store *any* relevant metadata as well as references to control and monitoring devices. The main concepts of the database together with their relations are represented on Figure 5.

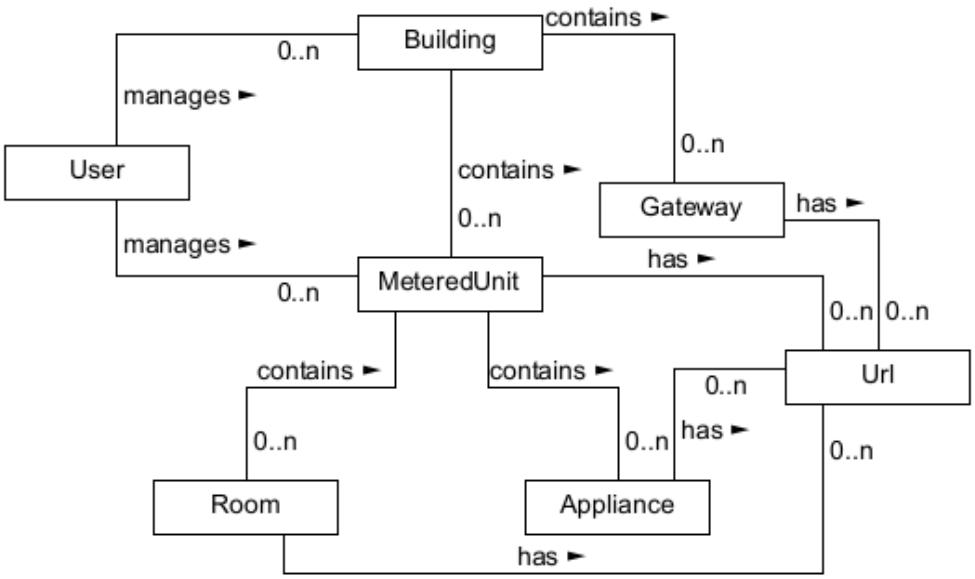


FIGURE 5: CONCEPTS OF THE DATABASE FOR METADATA

The main concepts are developed in Table 5.

TABLE 5: COMMENTARY FOR CONCEPTS OF THE METADATA DATABASE

Concept	Content
`Building`	Described building
`Gateway`	Hardware device hosting the cloud.iO endpoint
`User`	Person with the role of building owner or facility operator
`MeteredUnit`	Subdivision of a building whose consumption / production is measured by an official meter, typically an apartment
`Room`	Subdivision of a `MeteredUnit`, typically a room
`Appliance`	Piece of hardware that somehow acts on the building
`Url`	Base address for a piece of equipment providing monitoring and control data

The database stores URLs enabling the access to real-time monitoring data or to modify setpoints.

A URL stored within the `Url` table is the base URL of an online device or appliance. The type of device / appliance is also stored in the `Url` table. Provided the URL structure of a device / an appliance is known, the full set of URLs can be derived from the base URL.

The Figure 6 presents the structure of the `Url` table and Table 6 gives an example of an entry in the table.

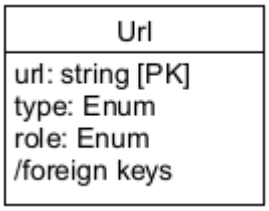


FIGURE 6: STRUCTURE OF THE URL TABLE

TABLE 6: SAMPLE ENTRY IN THE URL TABLE

Attribute `url`	Attribute `type`	Attribute `role`	Foreign key to « concept »*
`https://domos.oiken.ch/api/v1/data/cd03f4d7-4afb-4aba-830d-7a32b06a4b71/powerMeter-1`	`Aeotec Home Energy Meter Gen5`	`Measures Electricity`	*`An Appliance`*

The `url` attribute is the base url for the device / appliance. `type` indicates the model of the device / appliance, and `role` defines the role of the device / appliance related to the associated "concept" referred to by a foreign key.

An `url` attribute is the URL that an application can use to access a cloud.iO node. URLs for children objects and attributes – together with their meaning – are not stored in the metadata database. The node URL stored in the database serves as a prefix for the child elements, whose structure depends solely on the type of appliance / device stored in the database.

For example, https://domos.oiken.ch/api/v1/data/cd03f4d7-4afb-4aba-830d-7a32b06a4b71/power-Meter-1 is the base URL of a device of type "Aeotec Home Energy Meter Gen5". This device contains a few dozens of measuring points with, for each of them, a specific URL.

The design of the database assumes that not all properties of the concepts are known at design time. Hence, it should be possible to add new concepts on the fly, i.e., without modifying the database's schema. From a technical standpoint, extensibility is provided by associating a property table to each concept table (see Figure 7).

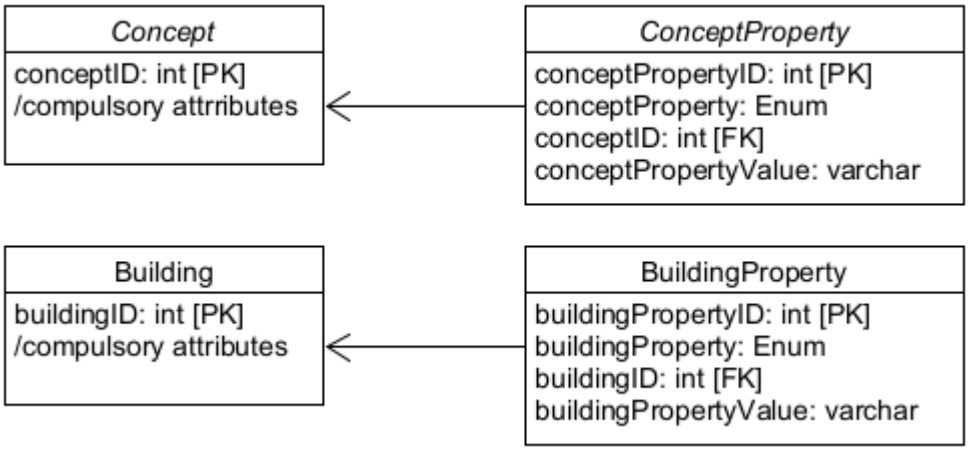


FIGURE 7: PRINCIPLE FOR DYNAMIC PROPERTIES

A property value for the building concept can be stored in the `buildingPropertyValue` of the `BuildingProperty` table by referencing the concerned `Building` and the concerned `buildingProperty`. The same pattern applies to other concepts. Adding a new property can be performed by adding a new entry in the `buildingProperty` enumeration.

The schema of the relational database is indicated on Figure 8.

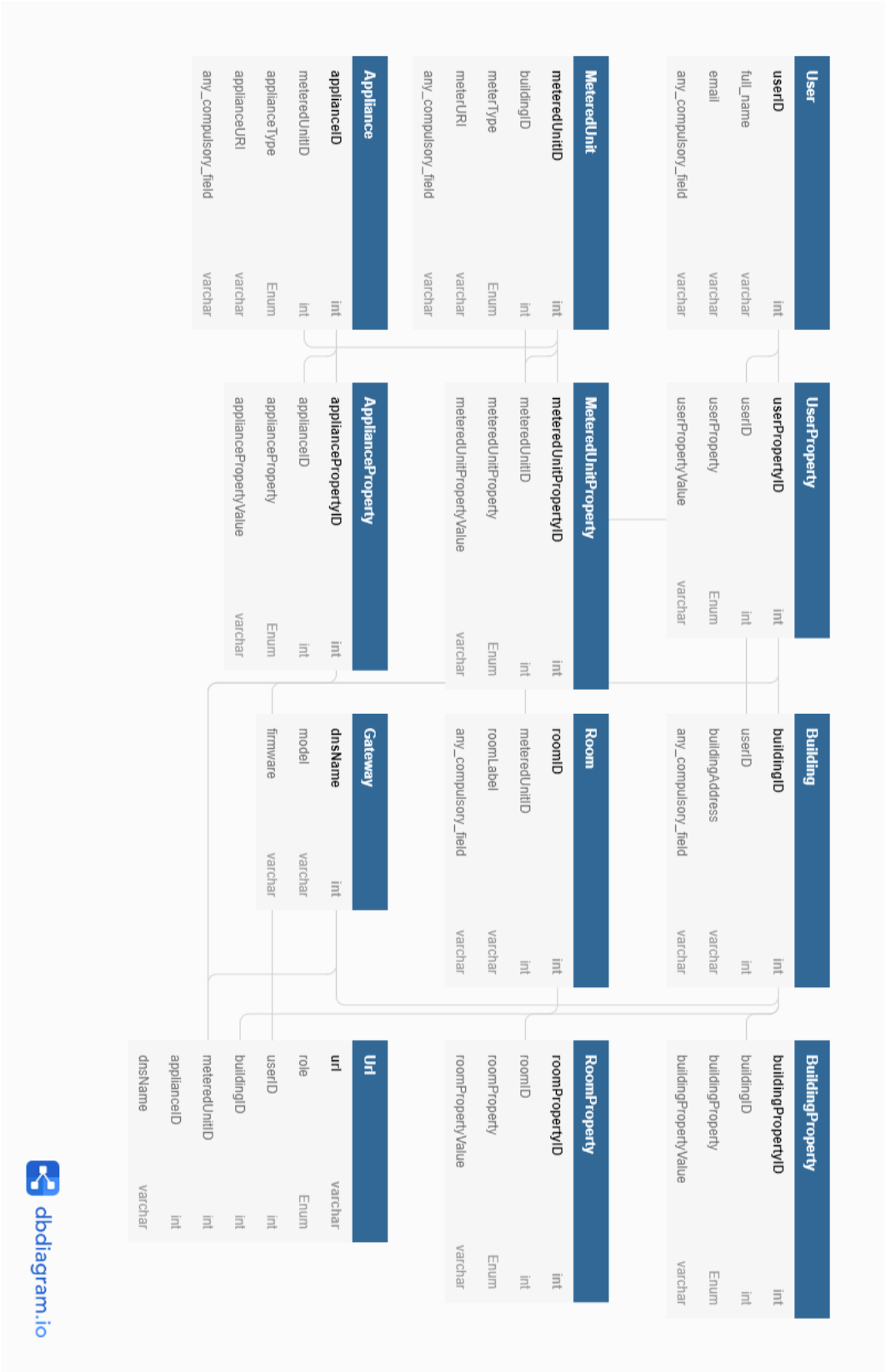


FIGURE 8: SCHEMA OF THE RELATIONAL DATABASE FOR THE DATABASE FOR METADATA

2.2. Description of Platform Implementation

2.2.1. Generation of BDs

The domOS IoT ecosystem as defined in (Nielsen, et al., 2022) requires that each participating building features a digital nameplate called Building Description (BD) containing relevant building's metadata related as well as also so-called interaction affordances. Interaction affordances, a key element of thee W3C's Web of Thing (WoT) architecture defined in (W3C, 2023) can be considered as a generalisation of the well-known concept of URL as it references any communication endpoint – and not only HTTP(S) endpoints. BDs are semantic documents compliant with the domOS Core Ontology (dCO)[1].

Information from two sources is required to craft BDs:

1. Metadata for the building as well as online appliances / devices with their role in the building stem from the metadata database.
2. Semantics is associated with the monitoring or control endpoints of involved online appliances / devices. The source of information is basically the manual of the appliance / device manufacturer.

BDs are generated by an ad hoc program when applications request them, see Figure 9.

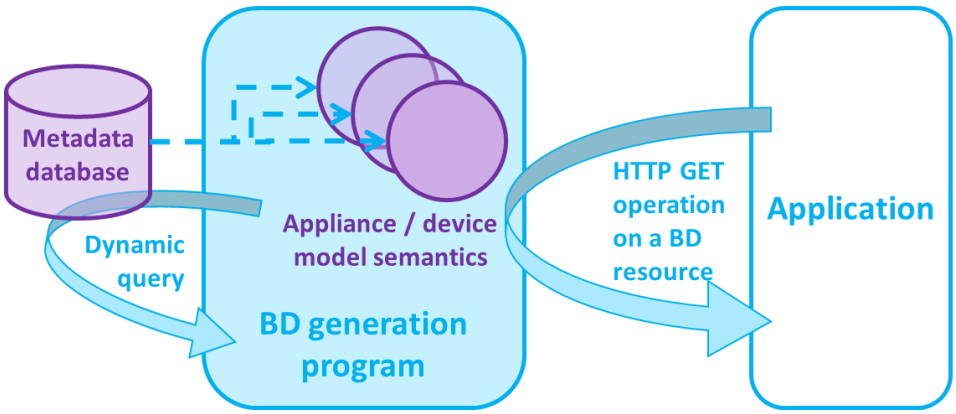


FIGURE 9: PROCESS TO GENERATE BDS

Admittedly, this approach generates well-formed and valid BDs, but the generating program is not generic and must be updated when new models of online devices / appliances are used. Looking back, the storage of raw BDs in a triple store would have been a better choice. The decision to use a relational database has been made before the definition of the BD concept and limitations appear later.

2.2.2. Generation of TDs

Thing Descriptions (TDs) are a key element of the WoT architecture. In their basic version, they describe the access to things'[2] instances. However, the TD specification (W3C, 2023)) also defines the Thing Model (TM) concept. A TM is basically a template for legacy TDs. It can contained placeholders for instance

[11] BDs are not described because the final version of the dCO and the present document are delivered at the same milestone.
[2] In this concept, a "thing" is an online device or appliance.

specific parameters (typically addresses) and can be transformed into legacy TDs by replacing placeholders with data that stem, in the case of cloud.iO, from the metadata database (see Figure 10).

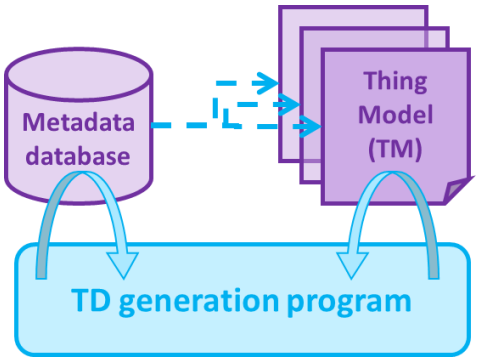


FIGURE 10: TD GENERATION PROCESS

In cloud.iO, a static TM is defined for each involved appliance / device model. TMs contains only cloud.iO's HTTPs URLs – and not native device / appliances interaction affordances.

2.2.3. Implementation

BDs and TDs are basically generated on demand when their URLs are called by a Python component made up of:

- a Flask HTTP server to distribute BDs and TMs on demand, and
- a BD generator program based on the RDFLib package. This package guarantees that generated BDs are well-formed RDF documents.

The cloud.iO node URLs contained in the metadata database are used to instantiate the TMs and are reported in the BDs. Note that cloud.iO attributes are mapped solely to WoT properties – the WoT events and actions concepts are not used. cloud.iO nodes, which model devices in domOS, may declare their compliance with an interface. In domOS, they do so by declaring the TM URL as interface identifier. This allows an endpoint to formally declare its compliance with a TM.

BDs must contain locators for individual properties, i.e., URLs cloud.iO attributes. As those are not present in the metadata database BD generating program must make use of information elements from the metadata database (queried dynamically) as well as of the static TMs (content hardcoded in the DB)

2.2.4. Reachability of the BDs

A cloud.iO platform dedicated to domOS is up and running in the OIKEN private cloud. A host acting as a BD and TD server is added to the above-described cloud.iO platform.

Applications are hosted in the same security zone as the BD server and have access to the BD / TD server. A static base URL serves the list of BDs (building an elementary building directory), identified by their URLs.

In the cloud.iO implementation, BDs can't be filtered: applications may access all BDs, which are only served in full. It is the responsibility of applications to parse BDs for locating the relevant information.

2.2.5. Reachability of the TDs

As indicated above, TMs – and not TDs – are hosted by a HTTP server as static documents. Parameters to transform a TM in a TD are provided as key-value pairs in BDs. Keys relate to placeholders in TMs, and TDs are generated by replacing keys/placeholders bay the corresponding values.

2.2.6. Description of Platform Security and Privacy Mechanisms

2.2.6.1. Type of Platform

In its current version, the cloud.iO platform is a closed platform according to (Nielsen, et al., 2022), even if cloud.iO platform could be operated as come an open platform. Using cloud.iO in a closed mode simplifies the deployment of applications while being compliant with the Sion demonstrator use case, where OIKEN oversees the in-building infrastructure, the cloud.iO platform and the applications.

The cloud.iO platform manages access rights for "entities". Entities can be applications or persons, typically identified by usernames and passwords, or web tokens. As of today, applications – and not persons (building owners / operators) – are registered as cloud.iO entities and therefore access rights are assigned to them. Persons are registered in the `User` table of the metadata database (see Figure 8), and all applications dealing with building owners / operators implement their own person access right management based on this table. This method enables a simple interface between an application and the platform, as only one account is required.

2.2.6.2. User Authentication

Users are defined as persons with the role of building owner, building facility operator, or just habitant. There is one and only one user account per building. Credentials for user accounts are stored in the `User` table of the metadata database.

Users access their account through a web service. User authentication is performed through HTTP basic authentication.

2.2.6.3. Application Authentication

Applications are registered in the cloud.iO platform and use HTTP authentication mechanisms.

2.2.6.4. Security

All endpoints and applications are connected to the platform using TLS authentication and encryption. Endpoints features their own X509 certificates and private keys.

Communication channels between applications and the cloud.iO platform as well as between web browsers of users and applications are secured using TLS.

2.2.6.5. Privacy

cloud.iO services allow assigning precise read / write access on endpoints' data model elements, down to the attribute level. As of today, each application is configured with similar access rights for all buildings.

Deletion of personal data – a GDPR requirement - is to be performed manually.

BDs are served unfiltered to demonstrators. This means that some applications could gain references to properties they may not access. This situation comes from the fact that the BD generator is independent of the requesting application and is therefore not aware of its access rights.

2.2.7. Description of Platform Management

2.2.7.1. Device Management

The cloud.iO platform disposes of a device (endpoint) management service. An endpoint may access the platform if it has a valid X.509 certificate and the corresponding private key. The endpoint management service allows generating sets of credentials (certificate and private key). A token is associated with each set. The token is entered on the endpoint at commissioning time. The platform features an open (i.e., without client authentication) HTTP service allowing the endpoint to fetch the set of credentials using an URL containing the token.

The association of device with a user is performed in the metadata database.

Devices are assumed to maintain open a MQTTs connection to the platform. The platform keeps track of the device status (online / offline).

Several methods for generating alarms or warnings upon device failure / disconnection are available: monitoring the status change of devices, monitoring the regular acquisition of key data, endpoint status registry managed by endpoints and published using cloud.iO. In the Sion demonstrator, alarms are generated when acquisition of key data is interrupted for a period longer than a given threshold.

2.2.7.2. Application Management and Enrolling

Applications are defined as cloud.iO users.

Commissioning a new application requires the following steps: 1) create the corresponding cloud.iO user, and 2) assigning read and write access rights to the application. Usually access rights are assigned on a set of similar cloud.iO nodes / objects / attributes in all buildings.

Applications needs an access to the cloud.iO platform, including the metadata database. No other condition must be fulfilled.

The life cycle management of applications is not part of the cloud.iO platform.

3. Implementation of the EDF Platform

3.1. Overview of the Platform

3.1.1. Architecture

The EDF platform is a scalable open-source IoT platform developed by the EDF Lab and based on the deployment of containerised Docker applications. The platform connects gateways deployed on the participating households' premises. A Zigbee coordinator is embedded in the gateways and manages a set of wireless local devices. The gateways communicate with the EDF platform via the participant's Wi-Fi router.

The EDF platform and the overall IoT infrastructure has been designed to comply with the EDF security policy. An example of requirement is that the platform must be installed on EDF premises.

The EDF IoT infrastructure is made up of the following components:

4. a central system named "Backend IoT platform" designed to manage gateways on the field,
5. gateways running a ZigBee coordinator (Zigbee Cluster Library (ZCL)),
6. ZigBee end devices, and
7. services running either on the backend IoT or on the gateways.

As depicted in Figure 11, an application is a piece of software that implements technical or business functionalities. Each application is embedded in a Docker container built on the backend IoT platform. The platform hosts a repository with all the docker images eligible for a deployment on the field.

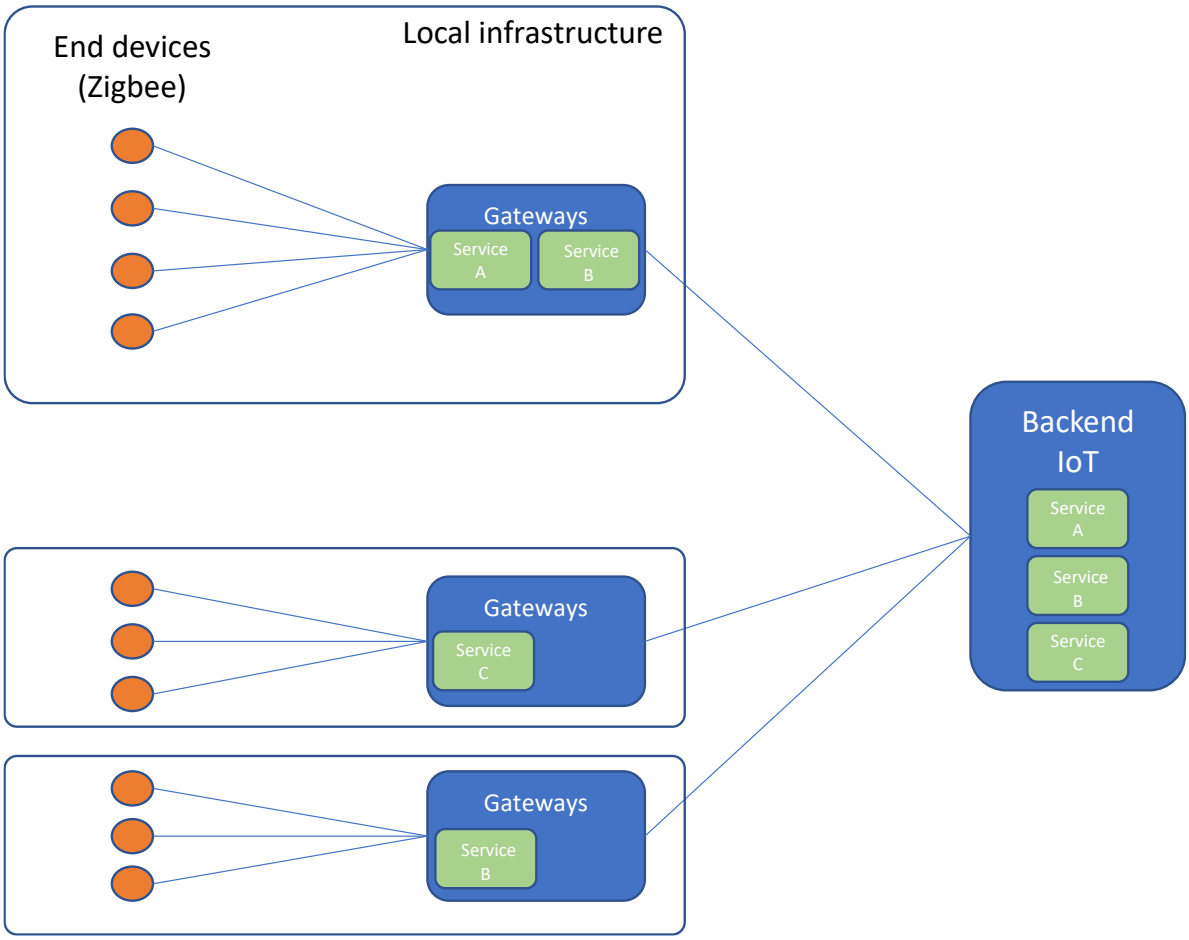


FIGURE 11: APPLICATION DEPLOYMENT IN THE EDF SYSTEM

The main components of the technical architecture are illustrated on Figure 12.

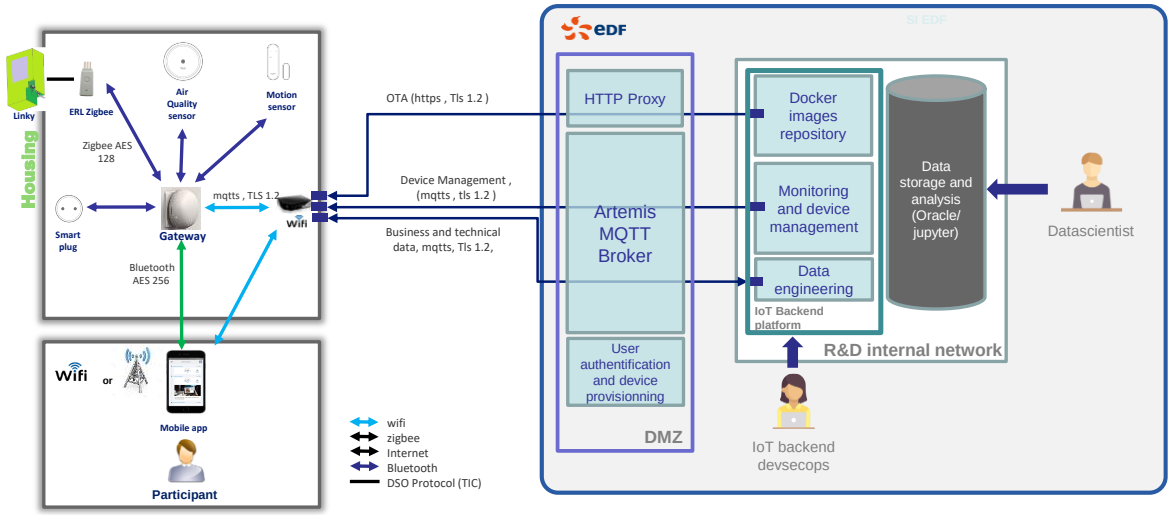


FIGURE 12: ARCHITECTURE OF THE EDF PLATFORM WITH LOCAL INFRASTRUCTURE

3.1.2. Implementation

The main features of the EDF platform are summarised in Table 7.

TABLE 7: MAIN FEATURES OF THE EDF IOT PLATFORM

Architecture	Linux based architecture with docker containerised microservices	
Micro-service platform	Docker	
Programming language	JavaScript	
Main components	Relational database management system:	Oracle
	Time series database management system:	InfluxDB/InfluxData
	Message broker	Artemis
Deployment version	VM, GitLab	

The gateway runs specific services on top of the firmware:

- A service designed to communicate with the Zigbee end devices. It runs the coordinator for the Zigbee network and hosts an open-source implementation of the ZCL. The service is named “Zigbee connector”.
- A service designed for routing the data to the EDF storage system in real-time. It translates the Zigbee endpoints into MQTT topics and implements a MQTT client to exchange data with the Artemis MQTT broker. The service is named “EDF Router”.
- A service designed for the installation of the EDF infrastructure in the participant’s house. The service maps Zigbee endpoints to GraphQL queries which are consumed by the mobile application specifically developed for the installation procedure. The service is named “EDF service” as shown in Figure 13.



FIGURE 13: MOBILE APPLICATION DISPLAY ON THE PLAY STORE

- A service specifically designed for domOS project, that translates Zigbee endpoints into TDs and BDs as specified in (Nielsen, et al., 2022). The service is named “domOS service”.
- A gateway with the capability to host other services that we will name “Business services”.

The services running on the gateway are illustrated on the software architecture schema of Figure 14:

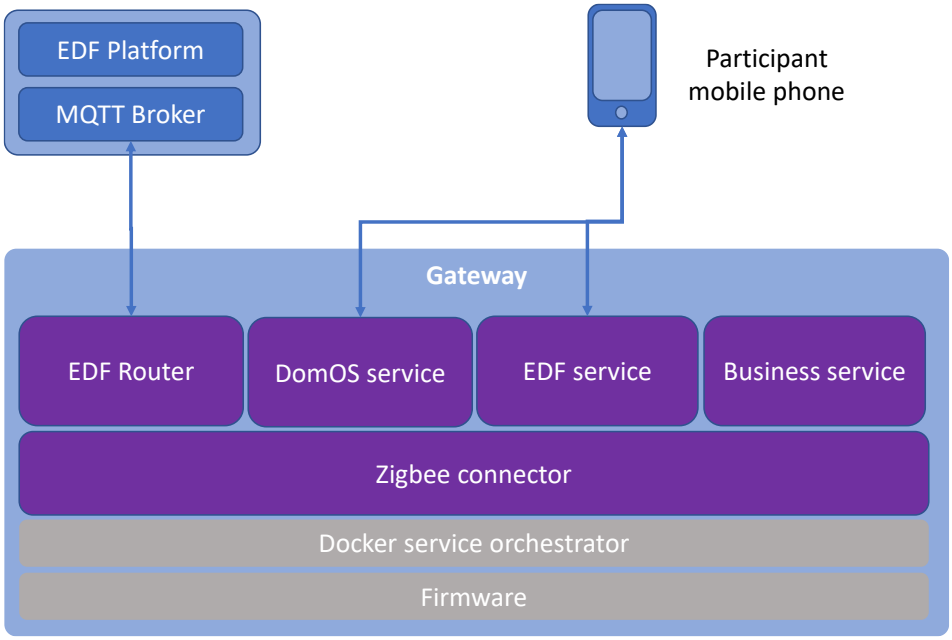


FIGURE 14: ARCHITECTURE OF THE EDF IOT PLATFORM

The gateway has also a storage capacity for historical data and user and device credentials.

Implementation features for gateways are summarised in Table 8.

TABLE 8: FEATURES OF THE EDF SYSTEM ENDPOINT

Language for endpoint library	Javascript/NodeJS, GraphQL
Communication library	Any implementation of ZCL, MQTT Paho

Gateways are connected to the message broker of the platform using the MQTTs protocol, with certificate-based authentication.

3.1.3. Zigbee

The “Zigbee connector” service hosts the PAN coordinator of the Zigbee network. Hence, it “owns” the network and can execute the following functions:

- starts / stops the network,
- opens the network for joining,
- allocates addresses,
- saves messages until they can be delivered (to its children),
- can function as trust centre.

The interactions between each type of devices in a Zigbee network and their data model are described in the ZCL specification (The ZigBee Alliance, 2019).

3.1.4. MQTT Topics and Data Model

Figure 15 illustrates the structure of MQTT topics of messages exchange by a gateway and the broker.

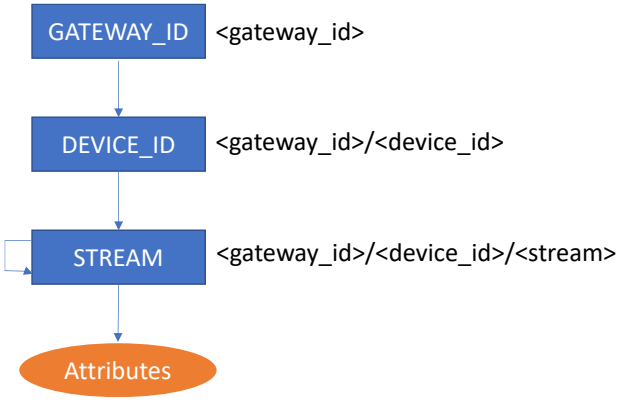


FIGURE 15: FORMAT FOR MQTT TOPICS

For example, the topic exposing the raw indexes of the ERL (Emetteur de Radio Local, Local Radio Emitter) is: `<gateway_uuid>/<erl_uuid>/<erl_indexes>`.

The corresponding payload is a JSON encoded message like the example presented on Figure 16.

```
{
  "date": "0000-00-00T00:00:00Z",
  "indexes": [
    123456789,
    123456789,
    123456789
  ]
}
```

FIGURE 16: SAMPLE JSON ENCODED MQTT PAYLOAD

3.2. Description of Platform Implementation

In the EDF pilot site, the domOS ecosystem is demonstrated at the local level, in a “edge computing like” architecture. The domOS service implements TDs and BDs, and links them with the Zigbee queries exposed by the Zigbee connector. Hence, it runs the abstraction layer on top of the Zigbee connector so that the Zigbee endpoints can be accessed through HTTP REST requests, in accordance with the TDs and BDs specification.

3.2.1. Reachability of the BDs

The BD is accessible through a static local URL.

3.2.2. Reachability of TDs

TDs are also accessible through local URLs, which are referenced in the BD according to the dCO.

3.2.3. Generation of the TDs and BDs

The BDs are created manually and integrated locally in the gateway.

For a maximal flexibility, each Zigbee device features its own TD.

3.2.4. Description of Platform Security and Privacy Mechanisms

The EDF platform is a closed platform as requested by the EDF security policy. Hence it has its own access management system and is hosted in the EDF network.

Users must be registered in the EDF authentication system.

Each gateway must be registered in the artemis MQTT broker registry. EDF operators can fine tune the access rights of each gateway to each MQTT topic.

Within the gateway, each service must use a token derived from cryptographic materials produced by the user's mobile phone and the gateway to consume the GraphQL queries.

3.2.4.1. User Authentication

The platform allows to fine tune their access rights among 3 generic roles: administrator, developer, and user.

- Administrator access rights are dedicated to users who operate the platform. They can configure the dashboards, grant access to other users, and manage services.
- Developer access rights are dedicated to users who develop services.
- User access rights are dedicated to users who need to have read only access to monitoring dashboard.

Pilot participants must download the Datalogger mobile application to provision the gateway and start the services. A gateway specific QR code is associated with each gateway. The mobile phone owner must scan the QR code to derive the credentials.

3.2.4.2. Application Authentication

Services are registered in a service registry embedded in the gateway. They are downloaded from the IoT backend platform through a public API that uses HTTP authentication mechanisms.

3.2.4.3. Security

The endpoints of the gateway (EDF router service) are connected to the artemis broker using TLS authentication and encryption.

The GraphQL queries published by the gateway uses TLS authentication and encryption mechanisms.

The user data are encrypted thanks to a private key derived from cryptographic materials from the mobile application and the gateway.

3.2.4.4. Privacy

All data stored in the gateway, user-related data as well as measurement data, are encrypted. If the gateway is stolen, confidentiality isn't broken.

The data sent to the EDF data lake are stored in a secured Oracle database.

Pilot participants can at any time send an email to a dedicated email box to claim his/her rights regarding his/her data. If the participant decides to quit the experimentation, a procedure to erase his data is triggered. The user receives a report form EDF once the data have been effectively deleted.

3.2.5. Description of Platform Management

The EDF platform runs a gateway management service. A gateway may access the platform if it has a valid X.509 certificate and the corresponding private key and is registered in the artemis MQTT broker registry.

The gateways are assumed to maintain open a MQTTs connection to the platform. The platform keeps track of the gateway and end devices status (online / offline).

3.2.5.1. Application Management and Enrolling

Commissioning a new service requires the following steps:

1) The developer builds the docker image.
2) The docker image is stored in the Docker registry of the IoT backend.
3) A Docker image download is triggered through a dedicated MQTT message sent to the gateway.
4) The gateway downloads the image containing the service and starts operation.

The life cycle management of applications and services is not part of the EDF platform.

4. Conclusion

The IoT platforms for the Paris and Sion demonstrators show many similarities (local gateways with peripheral devices, MQTT connection to the central platform). However, they chose a different approach for the implementation of the domOS ecosystem specification: the EDF platform focus more on peripheral “edge” computing serves BDs and TDs locally in each gateway, whereas the cloud.iO platform, featuring a more central architecture, implements BDs and TDs generation in the cloud. This is an illustration of the capability of the domOS ecosystem specification to address multiple design choices of IoT platforms.

5. References

Nielsen Brian [et al.] D2.3 Functional Specification of the IoT-Ecosystem [Report]. - 2022.

The ZigBee Alliance Cluster Library Specification [Online] // The ZigBee Alliance. - 2019. - https://zigbeealliance.org/wp-content/uploads/2021/10/07-5123-08-Zigbee-Cluster-Library.pdf.

W3C Web of Things (WoT) Architecture 1.1 [Online]. - World Wide Web Consortium (W3C), 2023. - https://www.w3.org/TR/wot-architecture/.

W3C Web of Things (WoT) Thing Description 1.1 [Online]. - World Wide Web Consortium (W3C), 2023. - https://w3c.github.io/wot-thing-description/.