Project Acronym: **BIMERR**
Project Full Title: **BIM-based holistic tools for Energy-driven Renovation of existing Residences**
Grant Agreement: **820621**
Project Duration: **42 months**

DELIVERABLE D8.2
BIMERR Middleware prototype

Deliverable Status: **Final**
File Name: **D8.2 BIMERR Middleware prototype_prereview_merged.docx**
Due Date: **28/02/2021 (M26)**
Submission Date: **26/02/2021 (M26)**
Task Leader: **FIT (T8.2)**

Dissemination level	
Public	X
Confidential, only for members of the Consortium (including the Commission Services)	

This project has received funding from the European Union's Horizon 2020 Research and innovation programme under Grant Agreement n°820621

The BIMERR project consortium is composed of:		
FIT	Fraunhofer Gesellschaft Zur Foerderung Der Angewandten Forschung E.V.	Germany
CERTH	Ethniko Kentro Erevnas Kai Technologikis Anaptyxis	Greece
UPM	Universidad Politecnica De Madrid	Spain
UBITECH	Ubitech Limited	Cyprus
SUITE5	Suite5 Data Intelligence Solutions Limited	Cyprus
HYPERTECH	Hypertech (Chaipertek) Anonymos Viomichaniki Emporiki Etaireia Pliroforikis Kai Neon Technologion	Greece
MERIT	Merit Consulting House Sprl	Belgium
XYLEM	Xylem Science And Technology Management Gmbh	Austria
CONKAT	Anonymos Etaireia Kataskevon Technikon Ergon, Emporikon Viomichanikonkai Nautiliakon Epicheiriseon Kon'kat	Greece
BOC	Boc Asset Management Gmbh	Austria
BX	Budimex Sa	Poland
UOP	University Of Peloponnese	Greece
UEDIN	University of Edinburgh	United Kingdom
NT	Novitech As	Slovakia
FER	Ferrovial Agroman S.A	Spain
UCL	University College London	United Kingdom

Disclaimer

BIMERR project has received funding from the European Union's Horizon 2020 Research and innovation programme under Grant Agreement n°820621. The sole responsibility for the content of this publication lies with the authors. It does not necessarily reflect the opinion of the European Commission (EC). EC is not liable for any use that may be made of the information contained therein.

AUTHORS LIST

Leading Author (Editor)			
Surname	**First Name**	**Beneficiary**	**Contact email**
Tavakolizadeh	Farshid	FIT	farshid.tavakolizadeh@fit.fraunhofer.de

Co-authors (in alphabetic order)				
#	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1	Devasya	Shreekantha	FIT	shreekantha.devasya@fit.fraunhofer.de
2	Grass	Alex	FIT	alexander.grass@fit.fraunhofer.de
3	Schell	Philip	FIT	philip.schell@fit.fraunhofer.de

REVIEWERS LIST

List of Reviewers (in alphabetic order)				
#	**Surname**	**First Name**	**Beneficiary**	**Contact email**
1	Lampathaki	Fenareti	SUITE5	fenareti@suite5.eu
2	Kousouris	Spiros	UCL	spiros@suite5.eu
3	Nektarios Lilis	Georgios	UCL	gnl2@cornell.edu
4	Katsigarakis	Kyriakos	UCL	k.katsigarakis@ucl.ac.uk
5	Giannakis	Giorgos	Hypertech	g.giannakis@hypertech.gr

REVISION CONTROL

Version	Author	Date	Status
0.1	Farshid Tavakolizadeh, Shreekantha Devasya	15/01/2021	Defined table of content
prereview	Farshid Tavakolizadeh, Shreekantha Devasya, Alex Grass, Philip Schell	10/02/2021	Quality Check
1.0	Farshid Tavakolizadeh, Shreekantha Devasya, Alex Grass	26/02/2021	Refinements to address internal feedback from internal reviewers. Ready for submission to the EC

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

This deliverable reports the work done within *T8.2 - Design & configuration of Middleware for Information Exchange throughout Architecture* as part of the *WP8 - ICT System Integration, Testing & Pre-Validation*.

The middleware design reflects the requirements of the project associated with secure and standardized information exchange among the various BIMERR components. Respectively, the middleware is designated for identity and sensor data management in the BIMERR project. The identity management is to maintain user and application profiles and authenticate the identity of resource owners. The sensor data management is a collection of functionalities to extract, maintain, annotate, and expose sensor data for building data collection and residential profiling applications. The middleware follows a microservice architecture which can be customized and deployed tailored to the applications' needs. The BIMERR project deployed a central set of components on the cloud and various instances of components on low-powered gateway devices in pilot sites.

The design and implementation of the middleware were carried out according to the plan, starting from M10 all the way to M26. The initial and refined designs were documented consecutively within the D3.5 and D3.6 BIMERR architecture documents. The current document reports the current architecture and deployment as a result of several iterative design, implementation and testing cycles. As the project progresses with the pre-validation and pilot stages, the middleware may require future work in form of minor refinements. Such changes will be reported in the deliverables of *T8.3 - End-to-end ICT System Integration Testing & Refinement* i.e., D8.4 (M30) and D8.5 (M40).

1. INTRODUCTION

The BIMERR middleware is a collection of components intended for smooth integration of ICT systems within the BIMERR system. The concrete design of the middleware is based on the requirements gathered over the initial project phases as reported in D3.1. The initial and refined designs were reported in D3.5 and D3.6; the two consecutive architecture deliverables.

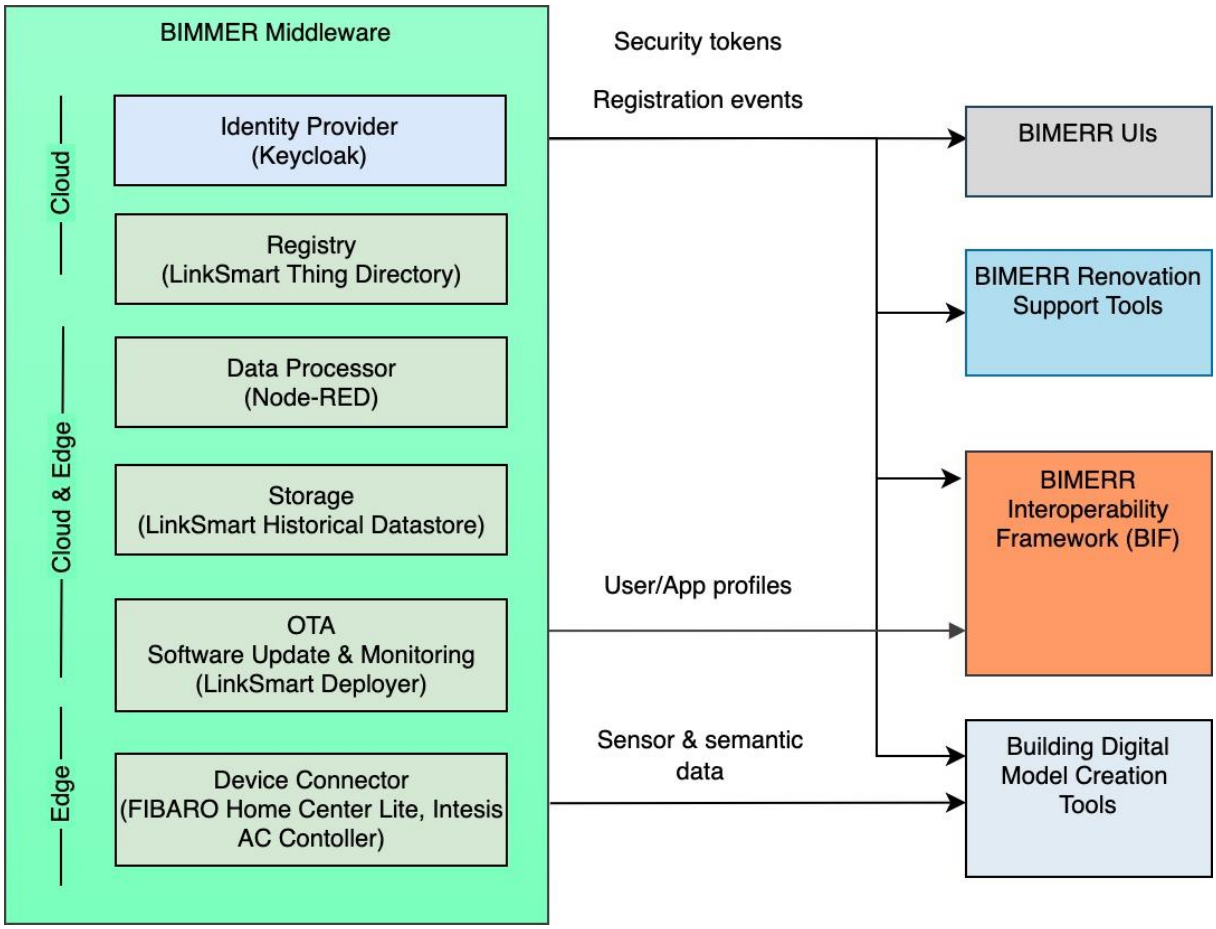

Figure 1. BIMERR middleware components and their interaction with other BIMERR sub-systems. The implementation names or acronyms are included in parenthesis. The middleware components are involved in identity management (blue) and data management (green).

In accordance with the project's requirements, the middleware focuses on two main tasks:

1. Identity management: A directory of users with roles and association to projects. The information shall be used to enable access control across the BIMERR system.

2. Data management: A distributed system to interface with the heterogenous wireless sensor network (WSN) to expose the relevant data in a structured, efficient, secure, and privacy-aware manner.

Figure 1 shows the components of the middleware and their interaction with the rest of the BIMERR system. The next chapters describe these components and the interactions in detail. The components, licensing, and distributions methods are listed in Table 1.

The characteristics of the middleware can be summarized as follows:

Modularity: The middleware is a modular system and customized to satisfy functional use case needs. The current design consists of six components. The *Identity Provider* is a central authentication server, a directory of user and application profiles, providing interfaces to authenticate them. *Registry* provides a searchable metadata directory for available devices (sensors, gateways), their endpoints and API (application programming interface) definitions. *Data Processor* mediates between various transfer protocols, routes live data, and transforms them to expected formats. *Storage* service provides time-series sensor data archival and retrieval capabilities. *OTA*[1] *Software Update & Monitoring module* allows remote software update and monitoring of gateway devices. Lastly, *Device Connector* provides abstraction on low level sensor protocols, exposing functionalities over higher-level IP protocols.

Efficiency: The components involved in data management place high focus on lowering the consumption of resources. This includes computational (e.g. CPU, RAM) and networking resources to minimize carbon footprints as well as costs. The approaches vary in different components, but generally involve processing close to the source coupled with economic data storage.

Portability: The components are packages as Docker images for various architectures. This allows deployment in the cloud or on the edge gateways (inside buildings or apartments) depending on the use cases. One goal is to offer most services on the producer side to reduce communication overhead and enhance data privacy. At the same time, it is possible to offer such services on the cloud, when physical constraints hinder local deployments.

[1] Over the air

Table 1. Summary of middleware component realization, licensing, and distributions.

Component Name	Implementation Name	BIMERR Extensions	Programming Language	Distribution	License
Identity Provider	Keycloak	None	Java	Docker images	Apache 2.0
Registry	LinkSmart Thing Directory	Fully developed within BIMERR	Go	Docker image, Debian packages, Binary distribution	Apache 2.0
Data Processor	Node-RED as the main programming tool	Application logic is fully developed in BIMERR	JavaScript, Python	Docker images	NodeRED: Apache 2.0 Node-RED flows and scripts: Proprietory
Storage	LinkSmart Historical Datastore	Synchronization between distributed storage nodes Storage backend optimization API Extensions: gRPC protocol for binary serialization and aggregation on-the-fly	Go	Docker images, Binary distributions	Apache 2.0
OTA Software Update	LinkSmart Deployer	None	Go	Docker image, Debian packages, Binary distribution	Apache 2.0
Device Connector	Fibaro Home Center Lite Intesis AC Controller	None	-	Pre-installed software on devices	Proprietory

Reusability: The middleware uses various open-source software in the implementations. In particular, it utilizes few components from the LinkSmart[2] platform and extends them in a generic way to satisfy the BIMERR needs while staying generic and reusable. Fraunhofer FIT is the main contributor to LinkSmart and these extensions are fed back to the platform as open-source contributions. The list of all reused components and BIMERR extensions are provided in Table 1.

Other quality attributes such as scalability, security, resilience, configurability, and reliability are discussed per component in the next chapters.

Chapters 2 and 3 describe the architecture of the two independent BIMERR subsystems for **identity management** and **data management** respectively. Those are followed by a brief overview of the middleware deployments in Chapter 4 supporting the BIMERR use cases. At last, Chapter 5 provides a summary of achievements and directions for future work.

[2] https://linksmart.eu/

2. MIDDLEWARE FOR IDENTITY MANAGEMENT

The identity management part of the middleware aims to provide the backbone to maintain user and application information and authenticate the owners of such data (resource owners) upon request. The requirement gathering and design of the identity management were performed as part of *T8.3 - End-to-end ICT System Integration Testing & Refinement* and reported inside *D8.3 - Integrated BIMERR ICT system 1*. In this deliverable we describe the component, called the Identity Provider, which realizes the expected functionalities.

2.1 IDENTITY PROVIDER

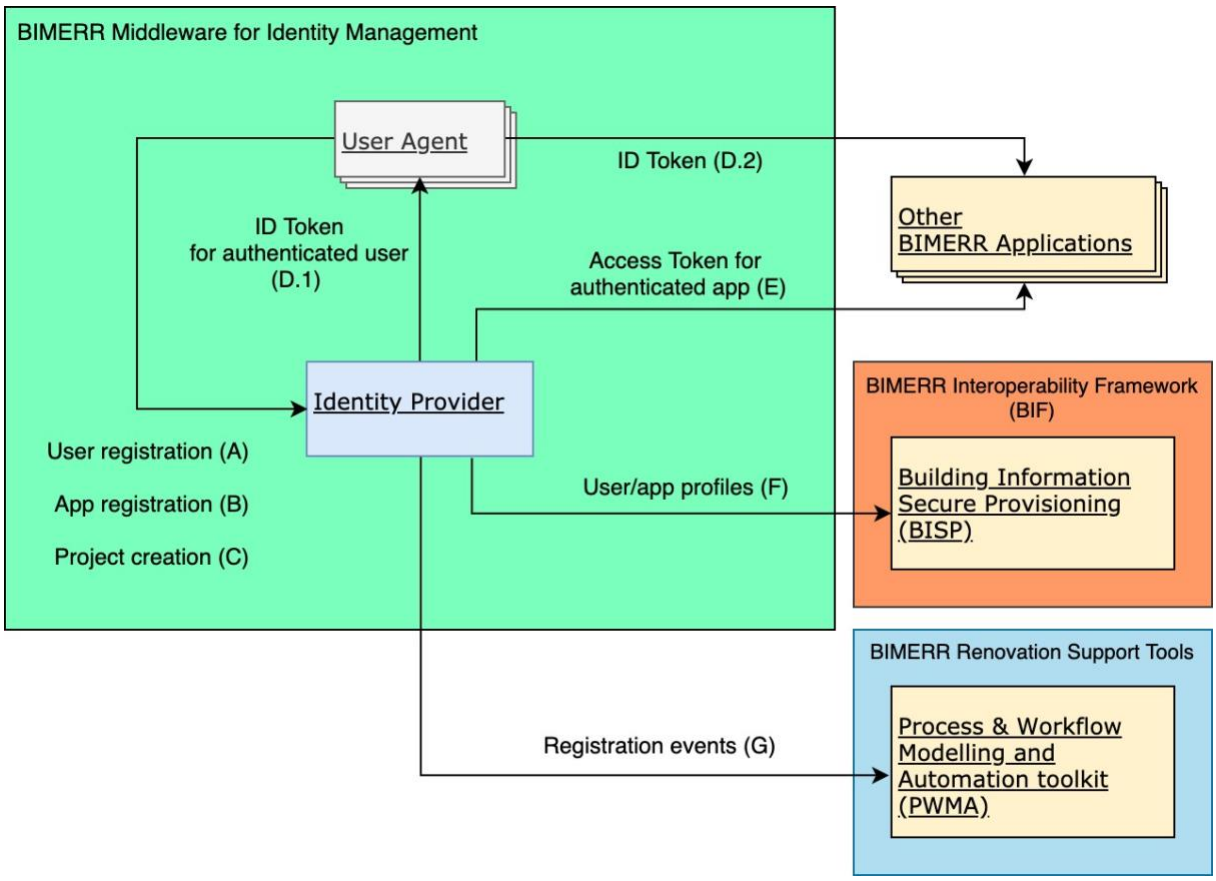

Figure 2. Data flow between Identity Provider and the rest of the BIMERR system. The labels do not indicate any specific order and are only used as reference in the descriptions.

The BIMERR Identity Provider is an authentication server implementing the OpenID Connect[3] protocol with additional APIs (application programming interfaces) for access management. We utilized Keycloak [4], an industry standard implementation for identity and access management. Keycloak is developed by Red Hat and other contributors and is available as open source under Apache 2.0 license[5]. It offers rich graphical user interface (GUI) and RESTful API[6] which are used to satisfy various identity management requirements of the project.

Figure 2 shows the data flow for identity management across the BIMERR system. At the centre, the Identity Provider interacts with other components in the following ways:

A. User registration on Identity Provider GUI via a user agent (e.g. web browser).
B. App (client) registration on Identity Provider GUI via a user agent (e.g. web browser).
C. Renovation/Construction project (user roles, user groups) creation on Identity Provider GUI via a user agent (e.g. web browser).
D. User log in and retrieval of identity tokens on Identity Provider GUI which can be passed to any BIMERR application as a proof of authentication when requesting resources. This is usually initiated with a redirect from an application.
E. Application authentication and retrieval of access token for requesting resources from other applications.
F. User and application profile retrieval for Building Information Secure Provisioning (BISP) component of BIMERR Interoperability Framework (BIF), responsible for enforcing access control on requests.
G. Registration events queries by Process and Workflow Modelling and Automation (PWMA) tool to detect necessary events and submit them to relevant users such as the BIMERR identity managers by email.

[3] https://openid.net/developers/specs/

[4] https://www.keycloak.org/

[5] https://github.com/keycloak/keycloak

[6] https://www.keycloak.org/docs-api/12.0/rest-api/index.html

The technical guides on how to use Keycloak APIs as BIMERR Identity Provider are available to project stakeholders. For the sake of completeness, these documents are provided in this deliverable as annexes:

- Annex A - Identity Provider: Token Request: information about authentication flows and token requests.
-

- Annex B – Identity Provider: Security Tokens: instructions about processing and validating the security tokens.
- Annex C – Identity Provider: Identity Provider REST API: a selected subset of useful Keycloak API endpoints and BIMERR examples.

Information related to using Keycloak's data model mapping to BIMERR and use of Keycloak GUI as a frontend for central BIMERR project metadata and user identity management are presented *T8.3 - End-to-end ICT System Integration Testing & Refinement* deliverable series (*D8.3-D8.5*).

3. MIDDLEWARE FOR DATA MANAGEMENT

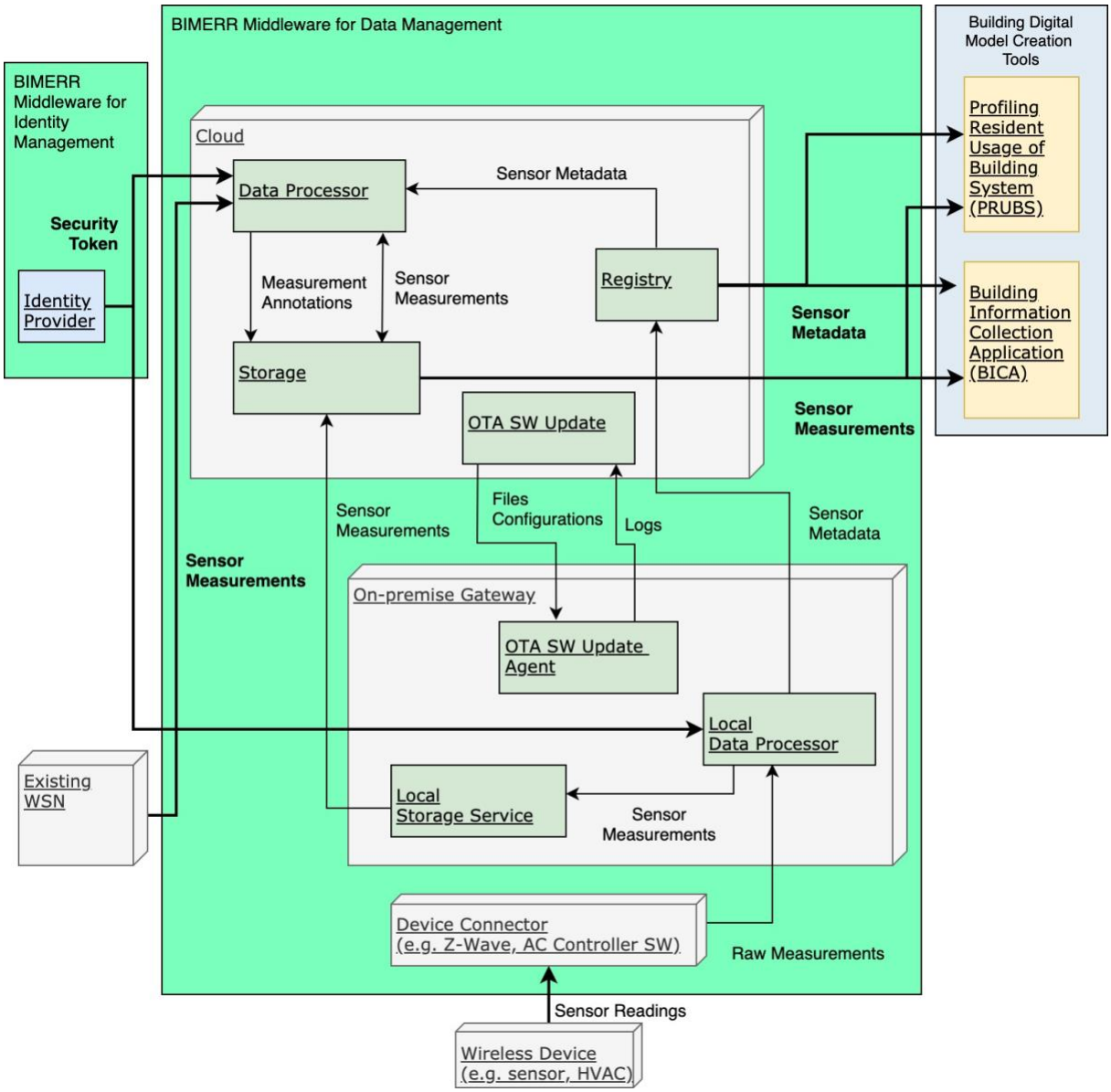

Figure 3. The overall architecture of middleware data management components with internal and external interactions.

The middleware consists of several components responsible for sensor data management. The high-level goal of these components is to retrieve data from the BIMERR wireless sensor networks (WSNs) and make them available in a structured and efficient manner to the BIMERR system. As mentioned in Chapter 1, one goal of the middleware design is to process and maintain most of the data close to the source and transfer only the necessary information to the cloud. This approach provides advantages such as optimal use of locally available storage

and computational capabilities, reduction of data transfer and cloud storage costs, simplification of ownership and privacy management. The disadvantage to this approach is that the locally maintained information may not be always reliable and available. The design of the middleware tries to address the challenges of making local computational resources more reliable and available; thus, ensuring that the advantages of the selected design outweigh the disadvantages. The data management components of the middleware along with the internal and external relations are illustrated in Figure 3.

3.1 REGISTRY

The BIMERR middleware operates on multiple interconnected networks consisting of numerous web services. These services provide software functionalities or give access to gateways and sensor measurements. It is possible to statically configure endpoints of services on the cloud. But this is not practical when it comes to services on edge gateways. The gateway services are often part of a dynamic network environment, making any form of static configuration obsolete very quickly. Networking solutions such as reserved DHCP[7] addresses or DNS[8] are not applicable since routing devices are beyond middleware's control. To be able to maintain such dynamic endpoints, we require a central registry with up-to-date metadata to be able to query and access required items.

Modelling of heterogenous devices is a challenging task. While it is tempting to develop a simple model which serves the project's requirements, there is always the risk of creating a model which is not extensible beyond the initial application. Moreover, the development of a new model reduces interoperability with other existing solutions. Among the available data models for device metadata modelling, two are standardized and most commonly used: OGC SensorThings[9] and W3C Web of Thing (WoT)[10]. The SensorThings standard is a data model and API (application programming interface) specification for sensors and actuators. The

[7] https://en.wikipedia.org/wiki/Dynamic_Host_Configuration_Protocol

[8] https://en.wikipedia.org/wiki/Domain_Name_System

[9] https://www.ogc.org/standards/sensorthings

[10] https://www.w3.org/WoT/

specifications are appropriate for centralized storage of both metadata and sensor data but fails to address situations where data is maintained in decentralized locations. Moreover, SensorThings enforces a single data model for measurements from all devices. The BIMERR system middleware needs to store data in a decentralized architecture and allow interoperation of heterogenous devices with various data models. The Web of Thing standard introduces the **Thing Description: a data model to describe metadata of devices along with the specification on how to interact with the device APIs.** The model is semantically annotated to provide interoperability with applications in other domains. Moreover, it can be extended to include domain-specific meta-attributes that may be needed to identify and interact with the devices.

The Web of Thing (WoT) standard was selected as it the most appropriate for the chosen architecture and the WSN devices used for residential profiling. At the time of making this decision, the WoT standard had no API specification for registration of Thing Description. BIMERR partners (Fraunhofer FIT and UPM) which are active members of the W3C WoT heavily contributed to the standardization of the API specification as part of WoT Discovery working group. The first public working draft of the WoT Discovery is now complete and available to the public (Cimmino 2020). The API of the BIMERR Registry (Directory of Thing Descriptions) was the basis for the initial draft of the WoT Discovery Registration API.

The implementation of Registry is in Go, as open source with Apache 2.0 and available as the LinkSmart Thing Directory[11]. This service exposes a RESTful API[12] over HTTP with endpoints to manage resources, as well as to search for particular ones, based on their properties. The API is protected from external access by means of Bearer Tokens obtained from the Identity Provider and access control based on OpenID profile of the requester[13].

[11] https://github.com/linksmart/thing-directory

[12] https://en.wikipedia.org/wiki/Representational_state_transfer

[13] https://github.com/linksmart/go-sec/wiki/Authorization

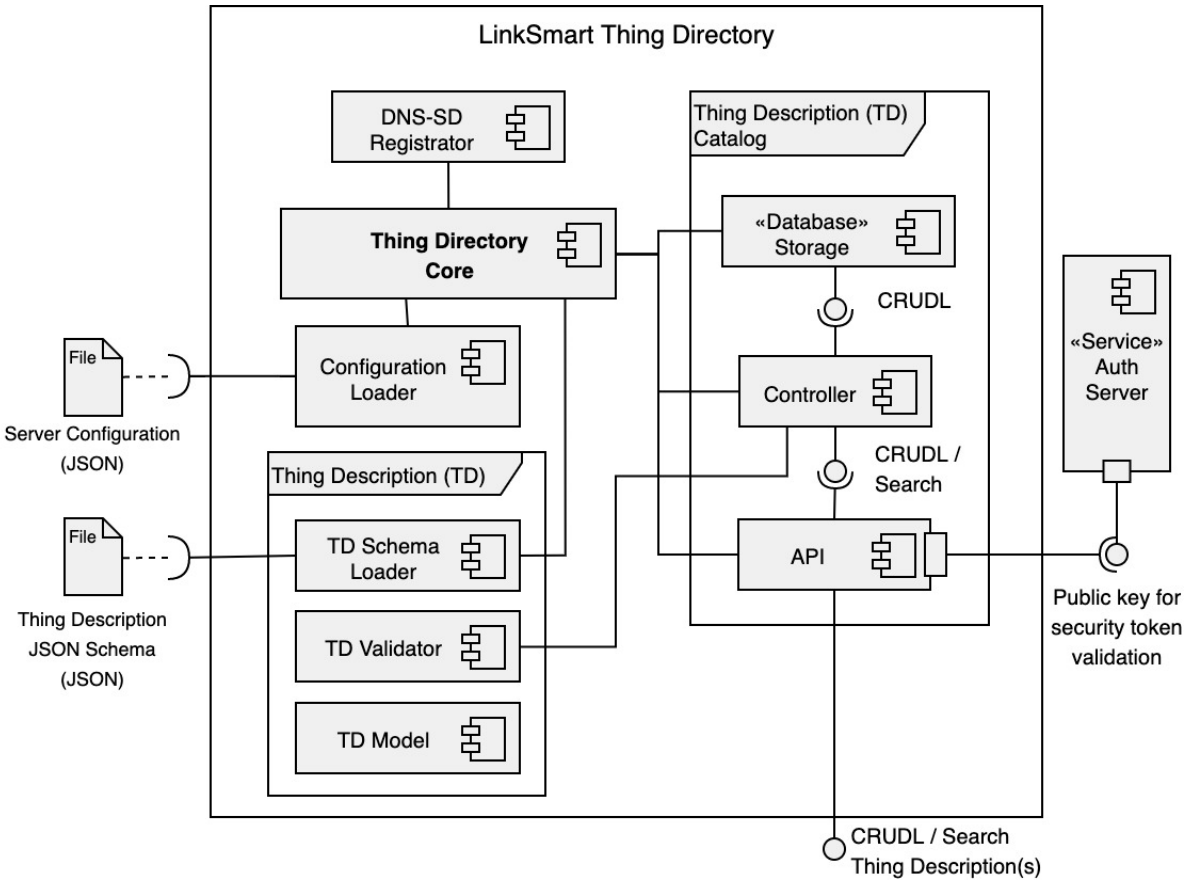

Figure 4. Component diagram of the Registry (realized as LinkSmart Thing Directory).

Figure 4 illustrates the components of Registry, the internal relations and exposed APIs. The **Thing Description Core** is the main component orchestrating all other activities. It interacts with the following components:

- **DNS-SD Registration** for service announcement in local networks. This is used for zero-configuration networking[14].
- **Configuration Loader** loads the server configuration file to setup the storage and API.
- **Storage**, **Controller**, and **API** are components of the **Catalog** package, responsible for maintenance and view of the Thing Descriptions. The **API** communicated with the **Auth Server** (BIMERR Identity Provider) to validate security tokens in requests.

[14] https://en.wikipedia.org/wiki/Zero-configuration_networking

- **TD Schema Loader**, **TD Validator**, and **TD Model** are components of the Thing Description package. As the names suggest, these components are responsible for loading of the specification and using it for validation of request bodies by the **API**.

The interaction between this component and the rest of the BIMERR system is illustrated in Figure 3.

3.2 STORAGE

The data collected from the sensors deployed at both pre-validation and validation sites is stored in a secure datastore to be available for the BIMERR services and applications such as Data Processor, BICA and PRUBS (see Figure 3). The storage solution is intended to run both on edge and the cloud. Availability of the sensor data in the edge is to support the analytics and aggregation services which run on the gateways installed at the residences. On the other hand, the data stored in BIMERR cloud is made available to the services which are deployed on servers away from the sites and do not have direct access to edge data to perform further processing of the transformed data. The data transformation happens at the various stages of the data processor (see 3.3). Storage also facilitates the querying of the data with varying sampling rates by aggregating the stored data.

There are plenty of storage solutions targeted for IoT scenarios. Apache IoTDB (Wang et.al. 2020) is an IoT native database with high performance ingestion and querying capabilities. Time series databases such as InfluxDB[15], Kdb+[16], TimescaleDB[17], Prometheus[18] are also studied and used in many IoT based solutions (Nasar et. al 2019, Shen et. al 2019). LinkSmart Historical Datastore on the other hand is a lightweight, distributed component for time-series IoT data storage supporting a standardized data format.

[15] https://www.influxdata.com/

[16] https://kx.com/platform/#timeseries

[17] https://www.timescale.com/

[18] https://prometheus.io/

BIMERR selected and extended LinkSmart Historical Datastore (HDS) as the storage component due to its lightweight nature and standardized data model. The core idea behind this component is to store data as close to the producer (apartment) as possible and only migrate the data toward fog (building level) and cloud when necessary. The component currently provides APIs for three protocols viz. HTTP, MQTT[19], gRPC[20] enabling request-response, publish-subscribe, and stream messaging patterns with support for JSON[21] and Protobuf[22] serializations. This enables efficient message exchange between the storage nodes, IoT sensor gateways, and consumer applications. The data model of the messages is based on Sensor Measurements List (SenML) (Arkko et. al 2018) specification, a simple and representative format enabling serialization of sensor data even on devices with very limited capabilities. The HTTP APIs are protected from external access by means of Bearer Tokens obtained from the Identity Provider with access control enforced based on requester's profile[23]; the MQTT API is protected by the MQTT broker; and the gRPC API is protected with mutual TLS[24]. APIs support multithreaded processing enabling high ingestion and query rates and scope for enhancing data storage backend to enable higher performance, consistency, and availability. The API also supports aggregation operations on the queried data enabling varying sampling rates. Storage is also integrated to open-source visualization tool Grafana with a Data Source plugin[25].

[19] MQTT is a standard for IoT messaging. https://mqtt.org/ provides more details.

[20] A open source Remote Procedure Call (RPC) framework. More details can be obtained in https://grpc.io/

[21] https://en.wikipedia.org/wiki/JSON

[22] https://en.wikipedia.org/wiki/Protocol_Buffers

[23] https://github.com/linksmart/go-sec/wiki/Authorization

[24] https://en.wikipedia.org/wiki/Transport_Layer_Security

[25] https://github.com/linksmart/grafana-hds-datasource

HDS is available as open source with Apache v2.0 License[26]. The documentation including deployment instructions and API specifications are available publicly[27]. Figure 5 depicts the component diagram of HDS. The main components are HDS Data and HDS Registry, responsible for providing APIs for sensor data management and sensor metadata management, respectively.

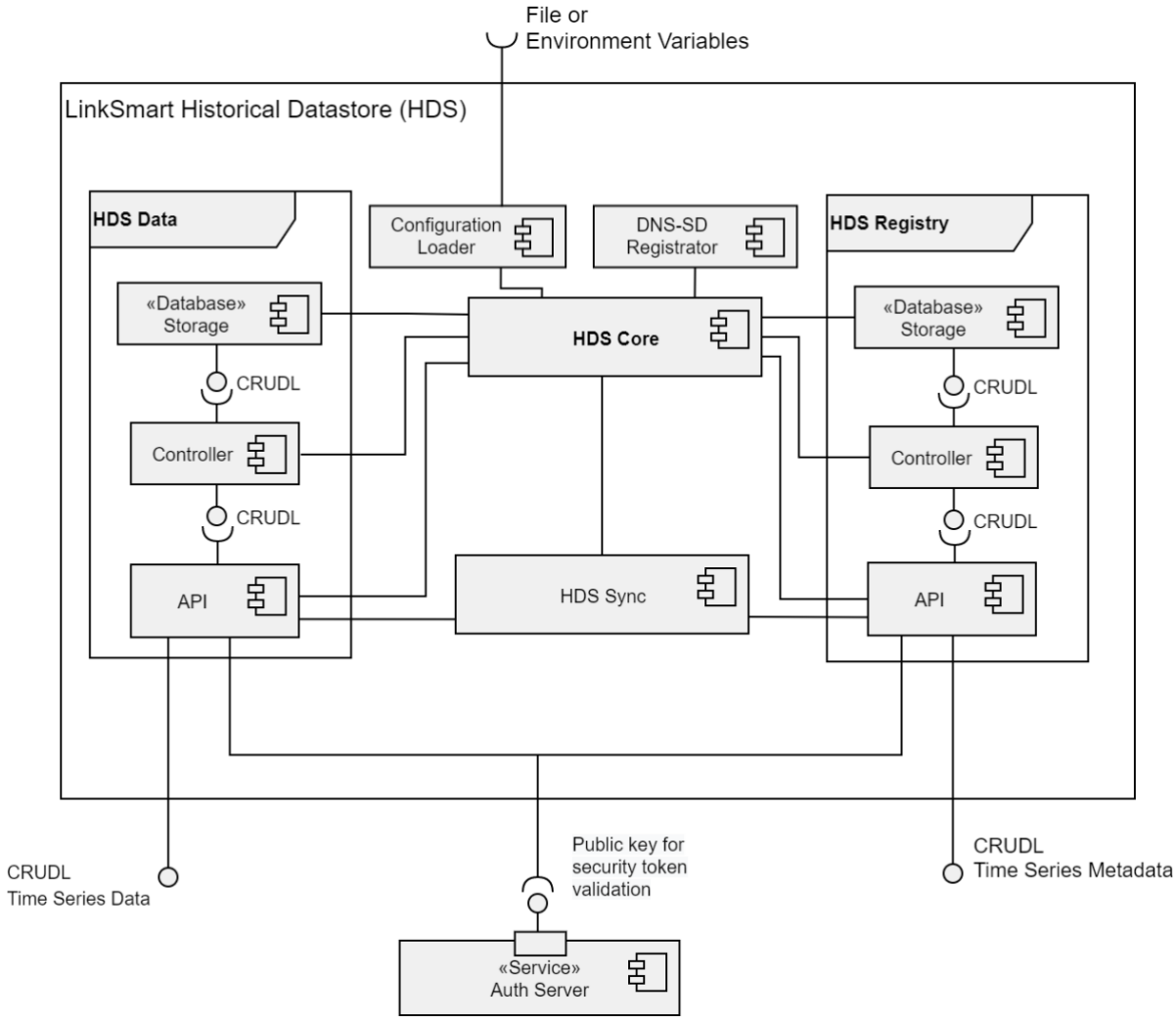

Figure 5. Component diagram of Storage (realized as Historical Datastore or HDS)

[26] https://github.com/linksmart/historical-datastore

[27] https://github.com/linksmart/historical-datastore/wiki

The main components of HDS are described below:

- **HDS Registry** is a package of components allowing registration and retrieval of metadata related to the time series data stored in HDS. The information includes storage configuration of the time series data. The API exposes HTTP and gRPC endpoints to interface with an underlying storage backend. The storage backend can be any storage system providing the required interface to the API component.
- **HDS Data** is a package which provides an interface to store and access historical time indexed data. The API exposes endpoints to store and retrieve raw data points. HDS Data exposes HTTP, and gRPC APIs supporting SenML media type. The API and media types are extensible. The storage backend is SQL-based and can be realized by any scalable SQL database. It also receives and handles notifications from the Registry API about changes to time series metadata and performs necessary actions to accept incoming data points from every source. This type of notification does not apply to storage backends that support built-in automated aggregation.
- **HDS Core** is responsible for setting up HDS and its APIs. The configuration is obtained from the *Configuration Loader* and is used to initialize required APIs and the OAuth Client. This component also takes care of the graceful shutdown of all components and service de-registration.
- **Configuration Loader:** This component loads and parses the JSON configuration of HDS and its APIs. These configurations can be overridden by environment variables. Upon start-up, the component loads the configuration object and provides an interface for structured retrieval of the attributes.
- **HDS Sync** takes care of synchronizing the historical data with another HDS instance. For instance, an HDS instance running on gateway enables synchronization component to synchronize its data with another HDS instance running on the cloud. This uses the configuration and the registry data to control whether a time series needs to be synchronized or not. This module uses gRPC APIs of both registry and data packages to perform synchronization.

3.3 DATA PROCESSOR

Data Processor is a combination of different services performing the data transformation and routing of sensor data. As shown in Figure 3, the data processor is deployed both on premise

and on the Cloud. It interacts with Device connector, Storage and Registry to achieve the following functionalities:

- **Onboarding:** Registering the devices in the BIMERR system by adding their meta data in the form of Thing Description in Registry.
- **Data Retrieval:** Extraction, transformation and routing of the sensor data and logs to the Storage.
- **Outlier Detection:** Detection of unusual patterns in the collected sensor data and annotation.
- **Alerting:** Alerting on unexpected events related to the sensor measurements.

These versatile functionalities of the Data Processor demand the capability to act on the data in near real-time. Hence an event driven processing platform is favored. Some of the event driven frameworks are: Eclipse Kura[28], Apache NiFi[29], Flogo[30], Node-RED[31]. Considering the ease of installations, availability of ready-made nodes for data processing, git integration and integrated visualization capabilities, Node-RED is chosen for partially realizing the Data Processor.

As shown in Figure 3, the Data Processor pulls the sensor metadata from the Device Connectors or existing wireless sensor networks (WSNs) along with static configurations to create Thing Descriptions as part of the Onboarding process. The Data Retrieval can function as soon as the Registry is populated with the device metadata. As part of the Data Retrieval process, the Data Processor performs required transformations of the data pulled from the Device Connectors and sends them to the Storage. Outlier detection currently happens as a post processing stage and works directly on the data stored in Storage. Alerting functionality reads the storage and creates alerts whenever there are events which needs the attention of maintainers. Data Processor fetches the token from Identity provider and uses this to

[28] https://www.eclipse.org/kura/

[29] https://nifi.apache.org/

[30] https://www.flogo.io/

[31] https://nodered.org/

authenticate itself to other middleware services. All the functionalities except Outlier Detection are realized within Node-RED programming environment. The different functionalities supported by the Data Processor are guided by the configuration files. The Data Processor exposes no public networking endpoints apart from a secure graphical user interface (GUI) for management.

The Data Processor is deployed across the cloud and local gateways. The Data Processor running on the cloud is responsible for onboarding and the transformation of the data coming from third party sources, as well as alerting and outlier detection. The Data Processor running on-premises (local gateways) takes care of onboarding and transformation of the BIMERR Device Connectors (more about Device Connectors can be found in Section 3.5). More details about the deployment are available in Chapter 4.

3.3.1 Onboarding

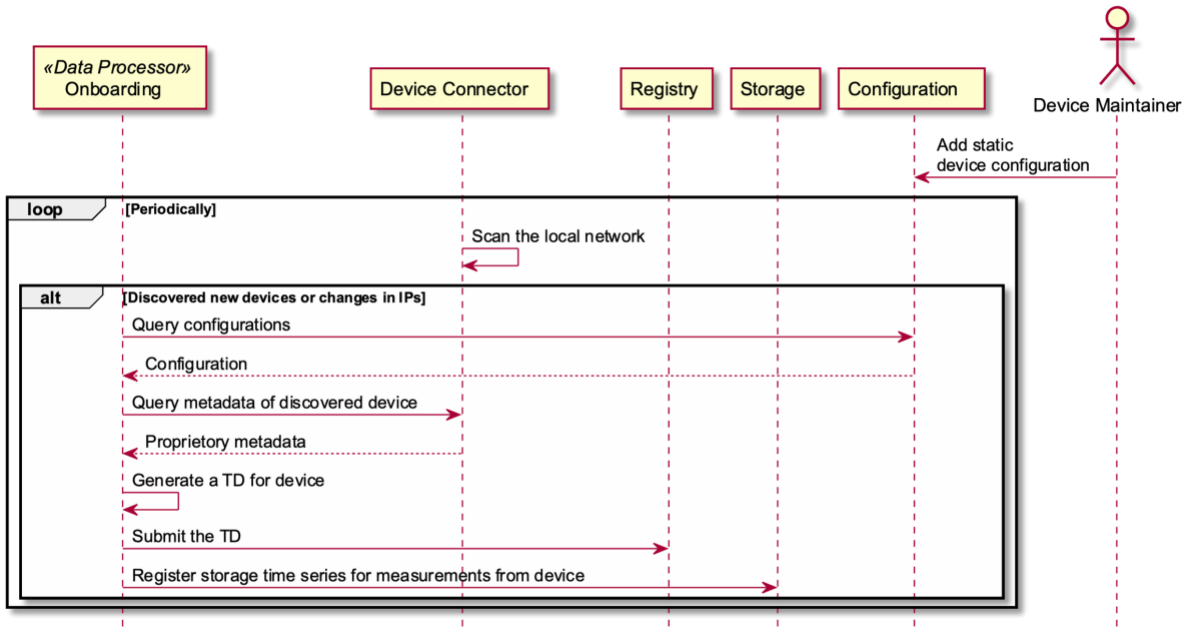

Figure 6. Data Processor's onboarding sequence

Onboarding is the process of connecting devices to the network and registering them for use inside a system. The Onboarding module is responsible for discovering newly connected devices and registering them into the Registry. Figure 6 shows the overall sequence from configuration by an operator until automatic registration. The initial state is the addition of device configuration by a human operator. These are typically static metadata about devices

stored in an online location (in this case a Git repository). The static metadata includes device-specific information such as types of devices, the interesting properties, and datatypes (See Annex E – Data Processor Sample Device Type Configuration). Moreover, the metadata includes project-specific information about each device, for instance the designated gateway, location data (apartment name, room, IFC Zone ID, IFC Space ID), and association with other local devices (See Annex F – Data Processor Sample Project Configuration for Devices). The Onboarding module queries the metadata and tries to discover devices periodically. When a new device is found, this results in a new registration. More often, the discovery results in changes in the IP address of existing devices and updates to registrations. The module then collects the information from the device and creates registration objects for Registry and Storage components. The metadata object for Registry is also cached locally to be used by other local Data Processor modules. The cloud components of the Middleware, as well as other BIMERR component query this metadata directly from the Registry. As explained in Section 3.1 (Registry), the chosen model is WoT Thing Description which includes metadata of a device along with the interaction details. An example of a Thing Description generated and registered by the Onboarding module is available in Annex G – Sample Thing Description for a wireless Luminance Sensor.

3.3.2 Data Retrieval

The Data Retrieval module fetches the data from the Device connectors, transforms it to the BIMERR specific format and routes it to the Storage. The overall sequence of the Data Retrieval is shown in Figure 7.

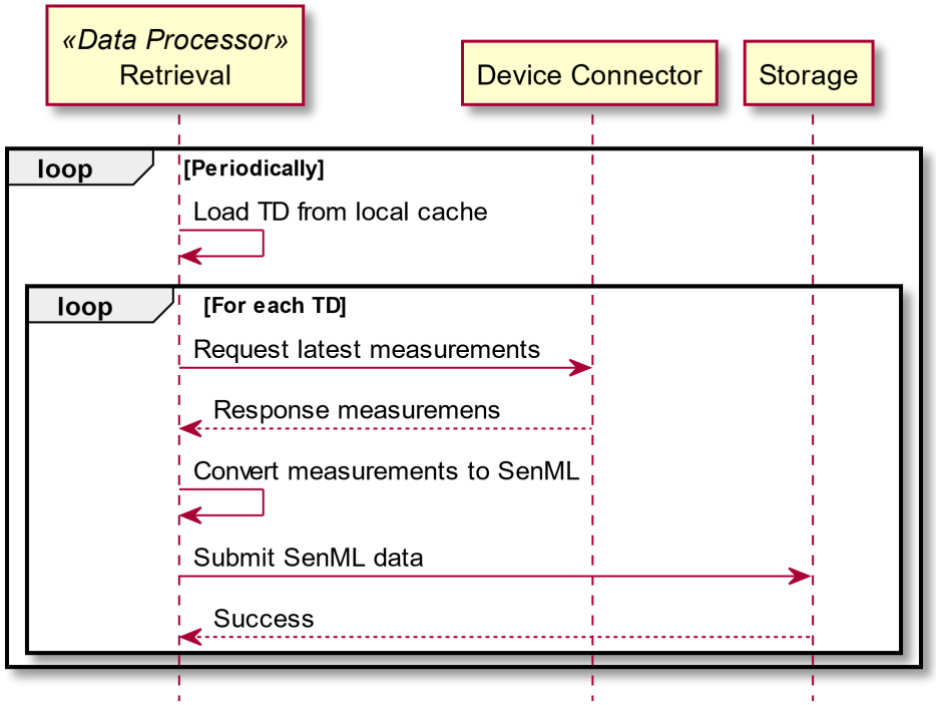

Figure 7. High level flow of data retrieval

The Thing Descriptions (TDs) are stored locally during the onboarding process which is re-used as part of data retrieval process. TDs have links to the interfaces exposed by the Device Connectors or other sources. The extraction operations vary based on the sources of the data:

- Z-Wave controller has a limited number of measurements locally. The data is retrieved using HTTP requests. Since the measurements are temporarily stored in Fibaro Home Center Lite (HCL), it is possible also recover the historical data in case of temporary failures of the connection between the data processor and the Fibaro HCL.
- Intesis AC Controllers are connected through TCP (Transmission Control Protocol) based connections. The instant measurements are retrieved in the form of ASCII messages exchanged through TCP ports. Since the AC controllers do not have internal storage, data will be lost whenever there is a connection interruption between the data retriever and the AC controller.
- Data from third party sources that is stored in an external server is fetched remotely by the cloud instance of the data processor.

The data retrieval from the various sources is performed at different rates depending on the type of the measurements. For example, the sensor measurements are fetched once in 10

minutes whereas the battery levels of the devices are retrieved only once a day (configurable). The collected measurements are first transformed to SenML format and then submitted to the Storage service.

3.3.3 Outlier Detection

Associated with the task of continuous data retrieval, ensuring its correctness is one of the major requirements for subsequent data processing. Especially in the context of IoT applications, where data is fetched from several heterogeneous sensors, it is essential to assure a predefined level of data validity. This becomes even more important when decisions are made based on machine learning models, which in contrast to the consideration of noise, often do not count in or work accurately for partially invalid data. The most prominent solution to this problem is Outlier Detection – often synonymous with Anomaly Detection. Since the data source is mostly assumed to work as intended, Outlier Detection is used to identify rare observations, which differ from the majority of the inspected data. Depending on the scenario, those outliers are to be equated with invalid data items. In the scope of BIMERR, Outlier Detection is used to annotate anomalies in retrieved sensor data (e.g., due to defective sensors), in order to improve data validity. Note that while there is no validation data available (e.g. labeled data), the resulting annotations are not reasoning about the related semantic or problem, but rather are an approximation based on the type of outlier, which needs to be validated by domain experts.

Outlier Detection is a highly researched topic, which is why the amount of research publications and thus the number of published solutions is immense. While most solutions tackle the more generic problem of Outlier Detection for tabular (time-independent) data, due to the context of sensor data, in this section only a brief introduction to Outlier Detection for time series data is given. In addition, in BIMERR no ground truth is provided with the retrieved data, which is why the number of approaches is further restricted to unsupervised ones.

When discussing Outlier Detection methods, most approaches can be grouped into the following categories: Probabilistic methods (Smyth et al. 1997, Aggarwal 2015), distance-based methods (Keogh et al. 2005, Christy et al. 2015) and prediction methods (Taylor et al. 2018). Probabilistic methods use a generative process or distribution to model the data. A common representative of the former technique, to probabilistically model values with a temporal dependency, is the Hidden Markov model. Apart from naïve distance-based approaches,

nearest-neighbor and clustering methods are well-known and more advanced candidates of this category. In contrast to probabilistic methods, they are regularly used for finding local outliers in time series data. At last, the idea of prediction-based methods is to model the behavior of a time series, such that deviations from the predictive model are regarded as outliers. Therefore, they are suited to find more complex outliers. Typical solutions of this category are regression models such as for instance autoregressive models. Despite of the approaches listed a good overview is given in (Aggarwal 2015).

Outlier Detection Architecture

While the problem associated with the aforementioned methods is frequently classified into three groups, namely Global Outlier Detection, Contextual Outlier Detection and Collective/Behavioral Outlier Detection, the approach in BIMERR is designed to be generic and in addition aims for stacking methods of those categories (Ensemble Outlier Detection). In order to allow for an individual processing of stored data streams from different sensors, the hereafter described solution consists of the following components:

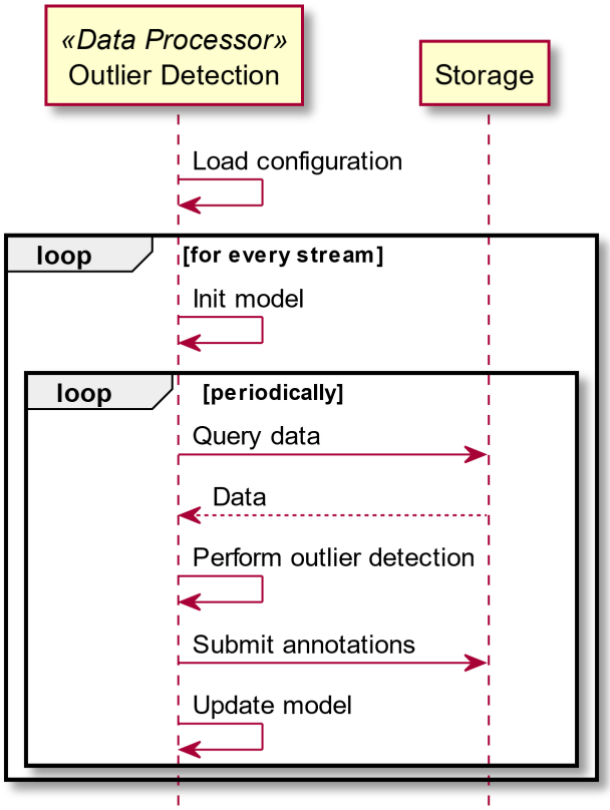

Figure 8. Outlier Detection: Core process flow

As one module of the Data Processor (see Figure 3), the Outlier Detection improves the validity of the stored data, by identifying and annotating outliers, in order to allow for a removal in subsequent data processing or analysis stages. Orchestrated by the App Manager and an optional REST Server, initially given a predefined configuration file, a Config Manager creates a Stream Processor for each listed data stream. Managed and executed by the Stream Processor, data retrieved with the help of the Storage connector is processed by one or more Outlier Detection algorithms in a parallel fashion, where the result is stored as an additional time series of annotations to the storage. Although this architecture allows for the inclusion of third-party Outlier Detection methods, the currently used Outlier Detection approach is a custom stacked/ensemble solution in order to capture different types of anomalies. The current choice of algorithms presented in the following section resulted from a first evaluation and the requirement of being comparatively adaptable/generic and not too strict in the assumptions in connection with a high number of different sensors. Methods in individual stages might therefore be adapted or replaced in future optimizations. Figure 8 summarizes the architecture and indicates the core flow of the processing components. All other components (with dotted boundaries) are rather used as resource or provisioning modules.

In the context of BIMERR, the outlier detection is executed in the cloud, where the collected data is incrementally fetched from the Storage and the individual results written into another annotation stream of Storage.

Outlier Detection Approach

Since invalid sensor behaviors are expressed differently within the data (unreasonable values, jumps, etc.), as a first step, global outliers are filtered out of the data by means of a fitted t-distribution. In contrast to an often-used normal distribution, the t-distribution is less sensitive to the majority of the data, such that sporadic deviations are not immediately defined as outliers. A point is regarded as an outlier if the distance from the mean is greater than a multiple of the standard deviation. As shown in Figure 9 (left) this first filtering stage eliminates entries, that do not match the learned distribution of the overall data.

Even when data points have a sufficient probability belonging to the valid population, they might still not have reasonable values according to their immediate local neighborhood. An example is given in Figure 9 (right). For this reason, DBSCAN (Density-Based Spatial Clustering of Applications with Noise) (Birant et al., 2007) is applied to a continuously sliding window of

the time series. This ensures that each point is only acceptable, if it is reachable from a predefined minimal number of points based on a maximally allowed distance. While respective deviations from the neighborhood are highly dependent on the context (e.g. a deviation of 1.5 could be unacceptable for values, which generally lie in the range of [0;2], but not for a range of [-10;50]), the approach exploits the fitted distribution from the previous stage, such that the allowed distance is a configured factor of the mean of the t-distribution.

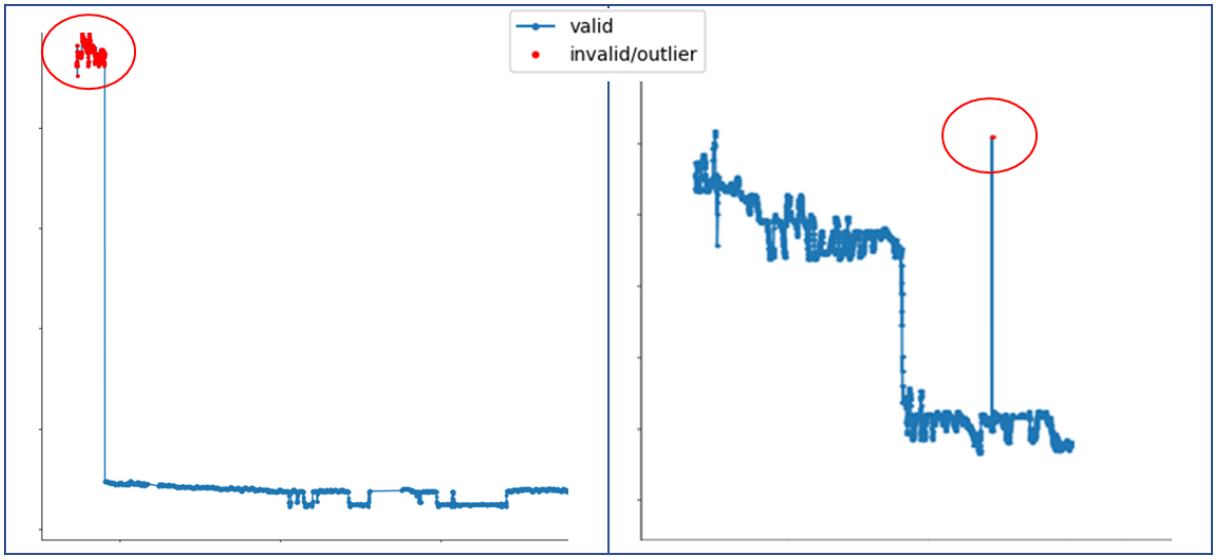

Figure 9. Global Outlier Detection (left) vs Contextual Outlier Detection (right)

While the global Outlier Detection in an online scenario is comparatively independent of the problem of dealing with missing data or sensor-based time gaps (e.g. due to energy saving modes), the local Outlier Detection depends on setting a reasonable neighborhood/window size. In future versions - also mentioned in the according section - one alternative solution is to skip local Outlier Detection for points that have no local context due to a lack of missing data in a predefined time window and to rely on the global solution.

As a last step, behavioral outliers, such as sudden changes in periodicity can be captured by the Behavioral Outlier Detection module. As this stage is not yet feasible, due to the currently limited amount of historical data and thus could lead to a potentially high number of false positives, it is not yet applied to the data, although the possibility of an integration is already given. One identified candidate in this context, resulting from literature research is

NeuralProphet[32], an open-source prediction library using neural networks. By creating an accurate prediction for the given time series, those predictions could be used to identify anomalies based on the deviation from the learned behavior.

Although the described default solution is used to identify outliers in BIMERR, each stage can be either added, skipped or replaced by newly developed or even more accurate solutions in future.

The implementation of the Outlier Detection was done in Python. Integrated third party libraries were Numpy, Pandas, Scipy and Scikit-Learn. The Outlier Detection module is currently a proprietary tool and it is envisioned to be open source in the future.

3.3.4 Alerting

Sensors and gateways deployed at the residential locations and the construction sites should ideally be operating without human interventions. In practice, the human intervention is inevitable considering the permanent device, network failures and unexpected events. Alerting helps identifying these situations. Alerting module of the Data Processor analyzes the data stored in Storage and triggers alerts whenever an unusual event is detected. The alerting module can run both in the cloud and on-premises.

The alerts are implemented as Node-RED flows. The alerts are generated at different intervals depending on the severity and the possibility of occurrence of the alerts. Alerts are not generated at all if there are no events related to it. Alerts are also deduplicated so that the alert subscribers are not overwhelmed with similar alert in a short period of time. The types of alerts currently enabled include:

- **Low battery levels:** whenever the battery level of a device goes below a threshold. The alerts are generated once a day.
- **Offline devices:** whenever a device or sensor does not produce data for long time. Since different sensors produce data at varying rates, this alert duration is configured based on the sensor types.

[32] http://neuralprophet.com/

- **Errors or exceptions in the gateway software:** whenever there is an unexpected error or exception in the gateway software. The errors are checked once in few hours and alerts are triggered whenever there is an exception. If there are more exception during this duration, the latest error is notified and with a link to navigate through the other errors.

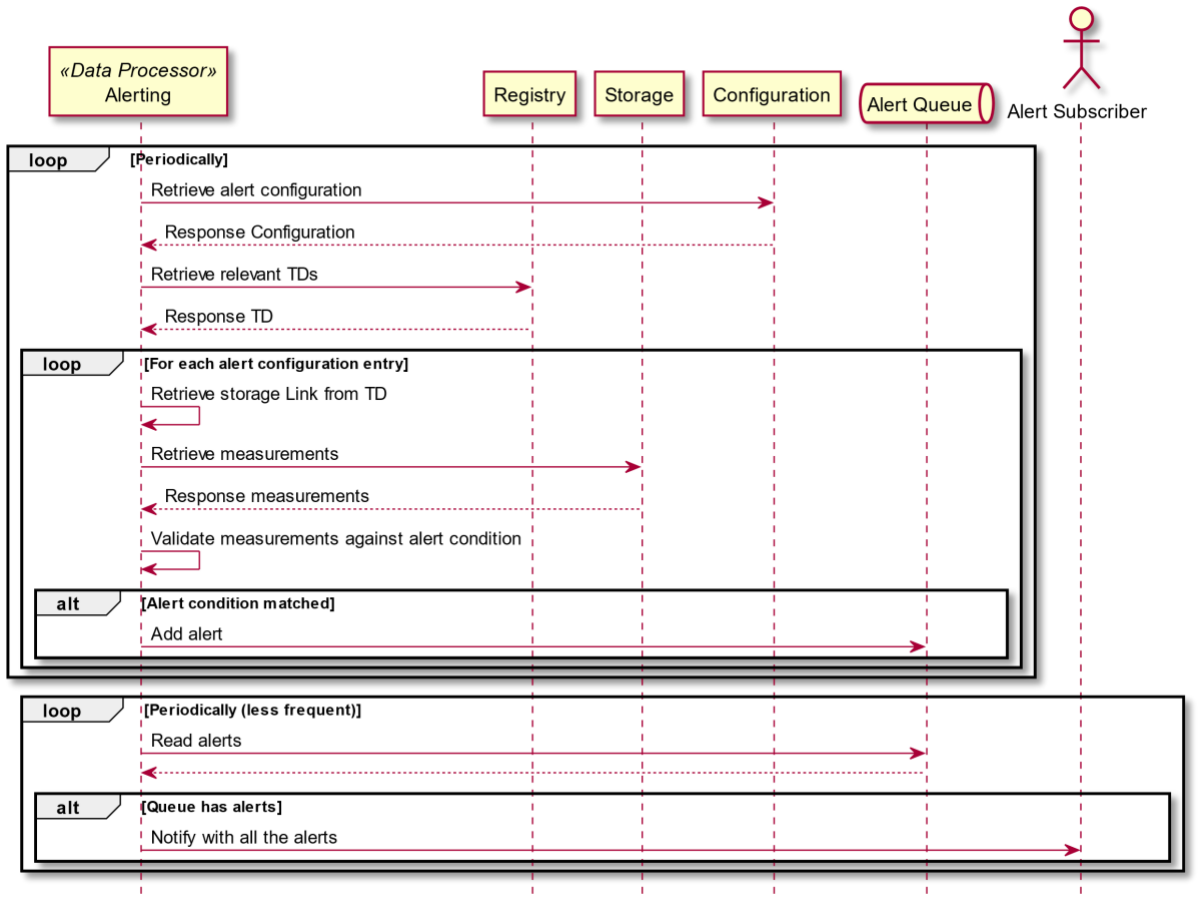

Figure 10. High level sequence for alerting

Figure 10 shows the overall sequence of alerting. Alerts are defined in a configuration file by the device maintainers (see Annex D – Data Processor Alert Configuration Sample). The Data Processor reads the configuration and loads the Thing Descriptions relevant to the defined alert conditions. The measurements that affect the condition are fetched from the storage and validation is performed. If an alert condition is matched, an alert event is generated. In order not to spam the Alert Subscribers with immediate alerts events, a controlled alerting mechanism is followed where an Alert Queue keeps all the alerts generated for a certain duration. The alerts later are combined to send to the Alert Subscriber. Data Processor also

ensures alert deduplication to avoid unnecessary alerts. An example device alert email is provided in Annex H – Sample Device Alert Email.

3.4 OTA Software Update and Monitoring

There will be an instance of the BIMERR middleware in every renovation site, integrating local services with other BIMERR components. These instances will be deployed on gateway devices with restricted connectivity and limited interfaces. The increase in the number of middleware instances and internal services is followed by additional complexity involved in the software provisioning. The OTA[33] Software Update and Monitoring component provides the necessary tooling and abstraction to simplify software provisioning in remote gateways. In particular, the component provides a graphical user interface (GUI) to the BIMERR middleware software maintainers to perform bulk software update operations and monitor the progress and runtime status of the components. This component is based on the existing LinkSmart Deployer[34] (Apache 2.0 license) and is being extended as part of the project. The decision to extend LinkSmart Deployer instead of other open-source solutions for software deployment and configuration (such as Ansible, Saltslack) was based on the architecture of those component as well as their usability. Ansible has an agent-less architecture and does not allow remote deployment on devices behind firewalls. Neither Ansible and Saltslack offer open source and free graphical user interfaces for deployment and monitoring of multiple devices.

This rest of this section presents the system design of OTA Update and Monitoring component. The architecture is identical to that of LinkSmart Deployer, developed mostly within the H2020 CPSwarm Project (Tavakolizadeh et. al., 2019).

[33] Over the air

[34] https://github.com/linksmart/deployment-tool

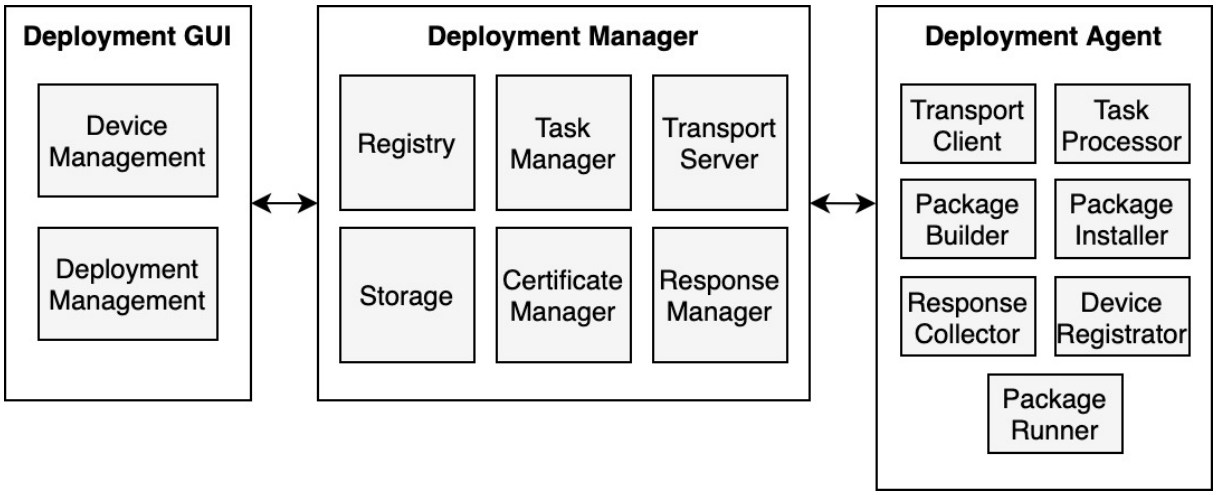

Figure 11. Components and internal modules of the OTA Update and Monitoring component (LinkSmart Deployer)

The OTA Software Update and Monitoring follows a client-server architecture with lightweight client components tailored for resource constrained environments, a scalable server component, and a graphical user interface to enhance the system usability. All components follow a modular design with low coupling and high cohesion; see Figure 11. This enables iterative development and maintenance of the system in a simple and structured manner. The server-side component, called Deployment Manager, is a centralized component with interfaces for user interaction and client communication. On the other hand, the client-side component, called Deployment Agent, runs on every device with very low footprints. The Deployment GUI is a web application that offers a graphical interface for the Deployment Manager and for simplifying various deployment operations. This section briefly introduces these components. The components are described below:

- **Deployment Manager:** This is the central component of the system with APIs to communicate with other components. Even though this is a centralized component, it is able to handle large amounts of traffic by vertical scaling. The Deployment Manager is developed with operation concurrency in mind in such a way that available resources are utilized efficiently during high load. Furthermore, the manager makes intensive use of queueing mechanisms to process high load of requests without causing congestion. This form of vertical scaling enables management of up to hundreds of targets. The system can further scale horizontally by instantiating multiple managers and load balancing at the API level.

- **Deployment Agent:** The client-side component of the system that runs on every target device (e.g. gateway). The design and implementation of the Deployment Agent place maximum focus on reducing runtime footprints. This is to ensure that the limited resources available on devices are kept available for other running application to the greatest degree. The Deployment Agent is mostly responsible for receiving tasks from the manager, validating and installing them, and afterwards managing their runtime lifecycle. A Deployment Agent could also be used to perform builds for other devices. Logging considerations at every step of the deployment assists developers in discovering deployment issues and adds transparency to remote devices.
- **Deployment GUI:** This is a web application that offers a graphical user interface for supporting various deployment operations. The user interface tries to tackle usability aspects of bulk deployment by introducing novel interaction methods in the whole deployment workflow. These involve intuitive ways to configure devices, roll out deployments, monitor the status and progress, and to diagnose issues more effectively.

3.5 DEVICE CONNECTOR

Device Connector is a component which adds interoperability to wireless sensor network (WSN) by abstracting the low-level communication protocols (e.g., z-wave, infra-red) and exposing interfaces over a TCP/IP network. The external interactions with this component are illustrated in Figure 3. The abstraction also includes serialization of primitive measurement values in a portable format such as JSON. The exposed interfaces enable setting and reading sensor and actuator configurations, as well as ways to read sensor measurements or set actuations.

The middleware does not implement any connectors from scratch, but instead relies on existing connectors or server software that comes with device controllers. The BIMERR deployments rely on three third-party connectors: Fibaro Home Center Lite and Intesis AC Controller. These connectors are described below.

3.5.1 Z-Wave Controller

The Z-Wave Controllers used in BIMERR project are Fibaro Home Center Lite (HCL). This device is an embedded computer with a pre-installed server software which provides APIs and a web-

based graphical user interface (GUI) for device management and data retrieval. FIBARO uses proprietary data models and APIs.

The Fibaro HCL is the connector and controller for the Z-Wave[35] devices (e.g. power meter, temperature sensor). The Z-Wave devices must be registered in advance and can only be managed from the controller interfaces. The controller also provides in-memory storage measurements collected from devices; this storage is capped at 10000 measurements.

Fibaro HCL provides a REST API to query sensor metadata and measurements. The official API documentation is available online[36]. However, the documentation is incomplete and, in few cases, invalid. A separate API documentation is served by the device itself, but that is also incomplete and partially valid. The necessary and valid API spec was developed by the middleware developers after inspecting the interactions between the GUI and APIs. This specification is available internally.

The Fibaro HCL allows configuration of data retrieval from Z-Wave devices. The configuration must be done with care and with respect to the user manual of individual device types. It is possible to set the reporting interval on the device itself. The frequency should be decided based on the device power source: battery-powered or a fixed power supply. More granular results will result from a shorter reporting interval, but this decreases the battery life considerably. A very short reporting interval would cause congestion when multiple devices are on the network.

It is worth noting that battery-powered Z-Wave devices sleep most of the time and report few times a day. This may lead to some inaccurate results, for example in case of temperature data, by missing out on temperature peaks such as when a window is opened briefly or sunshine for a short period.

[35] https://en.wikipedia.org/wiki/Z-Wave

[36] https://manuals.fibaro.com/knowledge-base-browse/rest-api/

3.5.2 Intesis AC Controller

Intesis Air Conditioner (AC) Controller interface gateways [37] are used to integrate air conditioners to the BIMERR system. The AC Controllers connect to the Heating, ventilation, and air conditioning (HVAC) devices through Infra-Red (IR) based connection and provide an interface to an IP network over WiFi.

The IP-based clients use TCP (Transmission Control Protocol) based communication on port 3310 where ASCII messages are exchanged. The monitored parameters include:

- power state of AC
- mode of operation: heat, cool, fan, dry, auto
- set temperature
- fan speed
- vane position
- ambient temperature
- error status of the AC
- error code of the AC

The AC Controller also provides interfaces for setting the values. The protocol specification of the controller is available in the product website[38].

[37] https://www.intesis.com/products/ac-interfaces/universal-gateway/universal-ascii-wifi-ac-is-ir-wmp-1?ordercode=INWMPUNI001I000

[38] https://cdn.hms-networks.com/docs/librariesprovider11/manuals-design-guides/wmp-protocol-specifications.pdf?sfvrsn=339b5cd7_6

4. DEPLOYMENT INFRASTRUCTURE

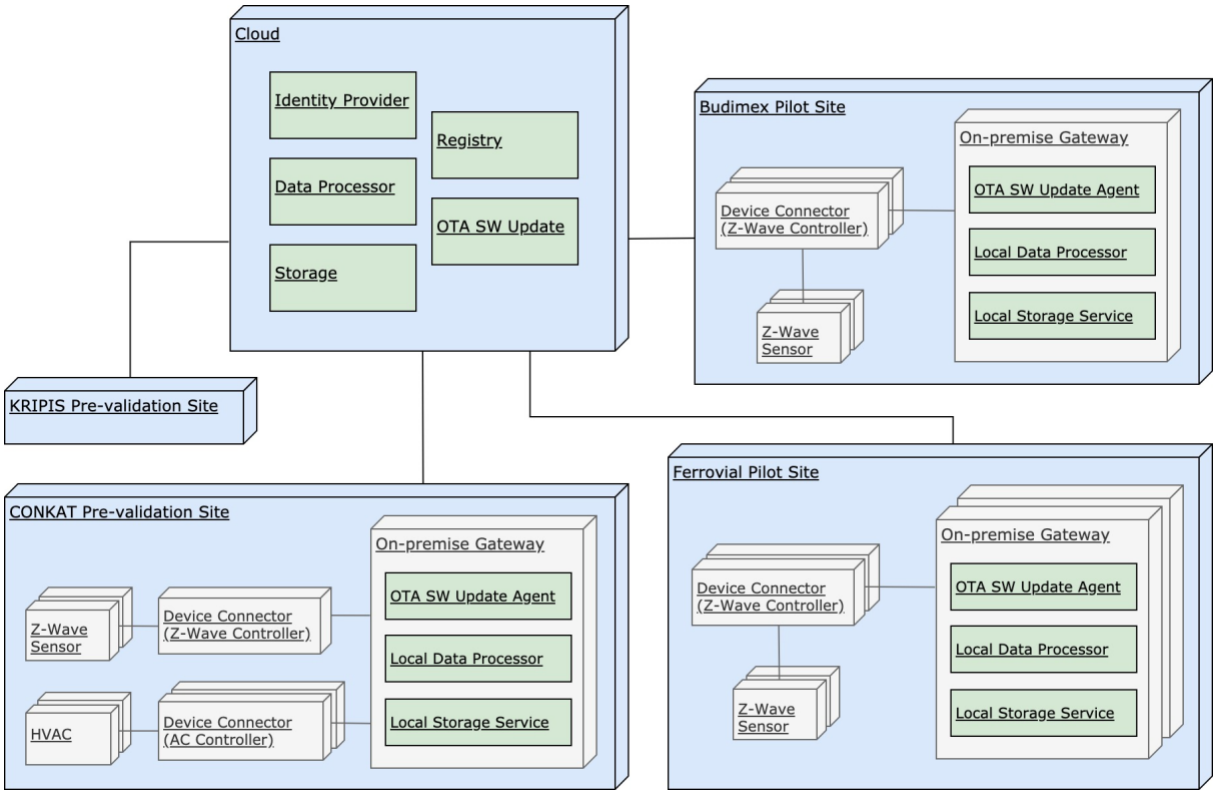

Figure 12. The deployment diagram of core middleware components (green), enabling sensor data management in pilot sites with varied characteristics.

The middleware is a collection of components deployed in the cloud or on-premises. The cloud components are centralized; the rest are pilot-specific and customized accordingly. Figure 12 shows the deployment model of middleware for BIMERR. The cloud components are deployed on an infrastructure provided by Fraunhofer FIT. These include the Identity Provider and Registry; the central Data Processor (with Outlier Detection Module), Storage, and OTA[39] SW Update components.

The cloud components are connected to four local deployments:

- KRIPIS Pre-validation Site: The KRIPIS smart home infrastructure by CERTH is a standalone system with various components. This site comes with a rich set of software

[39] Over the air

to query metadata and sensor measurements from cloud networking interfaces. The middleware relies on the interfaces provided by this platform to query meta and sensor data directly from the Data Processor. The details of the KRIPIS platform[40] are available online and beyond the scope of this document.

- CONKAT Pre-validation Site: The apartment with BIMERR wireless sensor network and gateway setup. This site consists of a single gateway (single-board computer) which hosts the local middleware components. The gateway connects to a Z-Wave [41] controller managing several Z-Wave sensors, and two AC Controllers paired with two Heating, ventilation, and air conditioning (HVAC) devices.
- Budimex Pilot Site: A building with a centralized gateway (single-board computer) which hosts the local middleware components. This gateway connects to several Z-Wave controllers, each handling communication to several Z-Wave sensors. At the time of writing, the execution of this deployment is pending.
- Ferrovial Pilot Site: A building with few gateways (single-board computers). Each gateway connects to a Z-Wave control responsible for several Z-Wave sensors. At the time of writing, the execution of this deployment is pending.

The wireless sensor network (WSN) is designed as part of *T5.4 - Profiling Residents Usage of Building Systems*. The final design and reasoning behind the setup will be reported in *D5.8 - Building resident energy related behaviour profiling framework 2*.

4.1 DEVELOPMENT TESTBED

A replicated setup of the wireless sensor network was deployed by Fraunhofer FIT as a development site to support middleware development and testing activities prior to the setup of the CONKAT pre-validation site. This setup consists of a single gateway (single-board computer), one Z-Wave controller, 17 Z-Wave sensors (13 wall plug power meters, one of CO

[40] https://smarthome.iti.gr/

[41] https://en.wikipedia.org/wiki/Z-Wave

Sensor, Door/Window Sensor, Motion Sensor, Danfoss Temperature Sensor). It is worth noting that most Z-Wave sensors include an additional sensor reporting the local temperature.

The development site helped in validation of the selected gateway, Z-wave controller's API and the middleware architecture before the start of pre-validation in actual site. Moreover, it helped driving the development of other components such as the Profiling Resident Usage of Building System (PRUBS) using the data made available at the earlier phase.

5. CONCLUSIONS AND FUTURE WORK

This document reported the current architecture and deployment of the BIMERR middleware. The architecture was designed based on project requirements and was validated with a testbed and the two pre-validation sites. While the overall architecture is expected to remain, the following improvements are foreseen in form of refinements to ensure the reliability of the wide range of functionalities throughout the project's validation phases:

- The **Registry** module is based on the W3C Web of Things (WoT) Discovery standard. The API (application programming interface) was initially in line with the standard, but the working standard is being developed to include new specification such as for notification and semantic search. These additions will be added to the Registry in the form of refinements.
- The synchronization capabilities of the **Storage** component are currently available and in operation. Future work may focus on simplifying the setup of the distributed storage system by adding an automatic configuration function.
- The Outlier Detection module of the **Data Processor** may be further refined to better generalize to a variety of sensor behaviours. Furthermore, the work will continue to distinguish false positives resulted during the installation of sensors or missed observations caused by sleeping battery-powered sensors.
- The **OTA Software Update and Monitoring** may be further tested in pilot sites and improved to enable visualization of the system status with more context awareness (e.g. building layout).
- The **infrastructure** in regard to pilot sites is tentative and subject to change depending on the available networking infrastructure. The middleware configuration model may need to change to avoid device identity collision in complex situation where gateways are shared among several apartments.
- The various middleware components provide APIs to remove user data. Such functionality shall be further exploited to provide building occupants the ability to remove the profiling data completely or for a specific period to protect their **privacy**.

6. BIBLIOGRAPHY

Aggarwal, Charu C. "Outlier analysis." Data mining. Springer, Cham, 2015.

Arkko, J., Keranen, A. and Bormann, C., 2018. Internet Engineering Task Force (IETF) C. Jennings Request for Comments: 8428 Cisco Category: Standards Track Z. Shelby.

Birant, D. and Kut, A., 2007. ST-DBSCAN: An algorithm for clustering spatial–temporal data. Data & knowledge engineering, 60(1), pp.208-221.

Cimmino, A., McCool, M., Tavakolizadeh, F., Toumura, K. Web of Things (WoT) Discovery. World Wide Web Consortium, 24 November 2020. The First Public Working Draft of WoT Discovery is https://www.w3.org/TR/2020/WD-wot-discovery-20201124/. The latest edition of WoT Discovery is available at https://www.w3.org/TR/wot-discovery/.

Christy, A., G. Meera Gandhi, and S. Vaithyasubramanian. "Cluster based outlier detection algorithm for healthcare data." Procedia Computer Science 50 (2015): 209-215.

Keogh, E., Lin, J., Fu, A. "Hot sax: Efficiently finding the most unusual time series subsequence." Fifth IEEE International Conference on Data Mining (ICDM'05). Ieee, 2005.

Nasar, M. and Kausar, M.A., 2019. Suitability of influxdb database for iot applications. International Journal of Innovative Technology and Exploring Engineering, 8(10), pp.1850-1857.

Padhraic, S.. "Clustering sequences with hidden Markov models." Advances in neural information processing systems. 1997.

Shen, L., Lou, Y., Chen, Y., Lu, M. and Ye, F., 2019, September. Meteorological Sensor Data Storage Mechanism Based on TimescaleDB and Kafka. In International Conference of Pioneering Computer Scientists, Engineers and Educators (pp. 137-147). Springer, Singapore.

Tavakolizadeh, F., Kiss, B., & Jánvári, B. (2019). D7. 4-FINAL BULK DEPLOYMENT TOOL, H2020 CPSwarm Project. Available at https://www.cpswarm.eu/wp-content/uploads/2020/06/CPSWARM-D7.4-v1.3_FINAL_UPDATE.pdf

Taylor, Sean J., and Benjamin Letham. "Forecasting at scale." The American Statistician 72.1 (2018): 37-45.

Wang, C., Huang, X., Qiao, J., Jiang, T., Rui, L., Zhang, J., Kang, R., Feinauer, J., McGrail, K.A., Wang, P. and Luo, D., 2020. Apache IoTDB: time-series database for internet of things. Proceedings of the VLDB Endowment, 13(12), pp.2901-2904.

ANNEXES

ANNEX A - IDENTITY PROVIDER: TOKEN REQUEST

ACCOUNT TYPES

There are two types of accounts:

- **User accounts:** for human users, where the human is the resource owner. The simplest form of authentication for user accounts is based on username/password.
- **Service accounts:** for services, where the service is the resource owner. Service accounts can authenticate using either of:
    - Client ID and Client Secret
    - Signed JWT
    - Signed JWT with Client Secret
    - X509 Certificate

GRANT TYPES

Keycloak implements various OAuth 2.0 grant types or flows. In is important to choose the right grant type depending on the application. This article provides a short introduction: https://dzone.com/articles/the-right-flow-for-the-job-which-oauth-20-flow-sho

AUTHORIZATION CODE GRANT

For traditional web apps, single-page apps, mobile apps, modern desktop apps.

More about this grant type: https://oauthlib.readthedocs.io/en/latest/oauth2/grants/authcode.html

From the front-end, redirect the user to:

```
<realm-endpoint>/protocol/openid-connect/auth?
client_id=bimerr-client&
response_type=code&
scope=openid&
redirect_uri=http://localhost:8080
```

Tip: during development and for testing, the `redirect_uri` may set to "urn:ietf:wg:oauth:2.0:oob" to get the `code` in the html body. But in production, `redirect_uri` should be set to the endpoint of your application to get the information from the query.

After a successful login, it redirects back to `redirect_uri` with two query parameters:

http://localhost:8080/?
session_state=<session-state>&
code=<auth-code>

The application can then use the `code` from the query parameter and request for a token:

```
curl --location --request POST '<realm-endpoint>/protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=authorization_code' \
--data-urlencode 'client_id=<client-id>' \
--data-urlencode 'code=<auth-code>' \
--data-urlencode 'redirect_uri=http://localhost:8080'
```

The response will include various tokens; see

Annex B – Identity Provider: Security Tokens.

RESOURCE OWNER PASSWORD CREDENTIALS GRANT

For trusted applications; user credentials are shared with the application and must be handled with care.

More about this grant type: https://oauthlib.readthedocs.io/en/latest/oauth2/grants/password.html

Example:

An application wants to provide access to a user in exchange for user credentials (rather than with tokens as usual). This must be avoided unless user interaction with a browser is impossible.

Request Tokens for Demo user (using token endpoint):

```
curl --location --request POST '<realm-endpoint>/protocol/openid-
connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=password' \
--data-urlencode 'client_id=<client-id>' \
--data-urlencode 'username=<username>' \
--data-urlencode 'password=<password>' \
--data-urlencode 'scope=openid'
```

The response will include various tokens; see

Annex B – Identity Provider: Security Tokens.

CLIENT CREDENTIALS GRANT

For machine-to-machine authorization when there is no user involvement.

More about this grant type: https://oauthlib.readthedocs.io/en/latest/oauth2/grants/credentials.html

Example:

An application wants to fetch data for internal use (not on behalf of a user) from another application.

Using `client_id` and `client_secret`:

```
curl --location --request POST '<realm-endpoint>/protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'client_id=<client-id>' \
--data-urlencode 'client_secret=<client-secret>'
```

The response will include various tokens; see

Annex B – Identity Provider: Security Tokens.

REFRESH TOKEN GRANT

For applications to continue to have valid tokens without further interaction with the user.

More about this grant type: https://oauthlib.readthedocs.io/en/latest/oauth2/grants/refresh.html

Using `refresh_token`:

```
curl --location --request POST '<realm-endpoint>/protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=refresh_token' \
--data-urlencode 'client_id=<client-id>' \
--data-urlencode 'refresh_token=<refresh-token>'
```

The response will include various tokens; see

Annex B – Identity Provider: Security Tokens.

ANNEX B – IDENTITY PROVIDER: SECURITY TOKENS

Server Issuer Endpoint:

https://auth.fit.fraunhofer.de/kc/realms/bimerr

A token response with scope=openid from Keycloak typically provides the following:

```
{
    "access_token": "<access-token>",
    "expires_in": 300,
    "refresh_expires_in": 1800,
    "refresh_token": "<refresh-token>",
    "token_type": "bearer",
    "id_token": "<id-token>",
    "not-before-policy": 0,
    "session_state": "<session-state>",
    "scope": "openid"
}
```

- **`id_token`** is a JSON Web Token (JWT) token that contains information about the user. To get the user profile, the application **must validate** and decode `id_token` programmatically.
- **`access_token`** may be used by an application to make an API request on behalf of the user.
- **`refresh_token`** may be used by an application to get a fresh set of security tokens on behalf of the user when the access and id tokens expire. This enables the application to continue to have valid tokens without further interaction with the user.

VALIDATION

Each `id_token` is self-contained and can be validated by the application without communication with the authentication server.

More information on how to validate: https://auth0.com/docs/tokens/guides/validate-jwts

Tip: server's public key can be queried from the issuer endpoint (link on top of the page). This is needed for signature validation.

DECODING

Online decoder: https://jwt.io/

Sample base64 decoded `id_token` <payload>:

```
{
  "jti": "<jwt-id>",
  "exp": <expiry>,
  "nbf": 0,
  "iat": <issued-at>,
  "iss": "<realm-endpoint>",
  "aud": "<client-id>",
  "sub": "<subject>",
  "typ": "ID",
  "azp": "<client-id>",
  "auth_time": 0,
  "session_state": "<session-state>",
  "acr": "1",
  "roles": [
    "Demo Role"
  ],
  "name": "John Doe",
  "groups": [
    "Demo Group"
  ],
  "preferred_username": "jane.doe@example.com",
  "email": "jane.doe@example.com"
}
```

ANNEX C – IDENTITY PROVIDER: IDENTITY PROVIDER REST API

The Identity Provider (Keycloak) REST API is accessible to admin users.

Documentation: https://www.keycloak.org/docs-api/6.0/rest-api/index.html

USEFUL ENDPOINTS

Description	Method + Endpoint
Get all users	GET /{realm}/users
Get all groups	GET /{realm}/groups
Get all realm roles	GET /{realm}/roles
Get all clients	GET /{realm}/clients
Get user's groups	GET /{realm}/users/{id}/groups
Get user's realm roles	GET /{realm}/users/{id}/role-mappings/realm
Get users of a group	GET /{realm}/groups/{id}/members

Get users with a role	GET /{realm}/roles/{role-name}/users
Get user events	GET /{realm}/events
Get admin events	GET /{realm}/admin-events

EXAMPLES

First, get an `access_token`:

- For your **admin user** using `bimerr-client` and via one of the oauth grant types (see Annex A - Identity Provider: Token Request). Since we need an access_token for realm administration (not id_token for user profile), the scope should be set to "roles" in the request (instead of "openid").
- For your **application**, use client credentials grant with a client that has appropriate admin roles.

  Example:

```
# Using client id and client secret:

curl --location --request POST '<realm-endpoint>/protocol/openid-connect/token' \
--header 'Content-Type: application/x-www-form-urlencoded' \
--data-urlencode 'grant_type=client_credentials' \
--data-urlencode 'client_id=<client-id>' \
--data-urlencode 'client_secret=<client-secret>' \
--data-urlencode 'scope=roles'
```

GET LIST OF USERS

```
curl --location --request GET '<realm-endpoint>/users/' \
--header 'Authorization: Bearer <access_token>'
```

Response:

```
[
    {
        "id": "<user-id>",
        "createdTimestamp": 1589290417657,
        "username": "demo@bimerr.eu",
        "enabled": true,
```

```
        "totp": false,
        "emailVerified": true,
        "firstName": "",
        "email": "demo@bimerr.eu",
        "attributes": {
            "description": [
                "This user is just for demo purposes"
            ]
        },
        "disableableCredentialTypes": [
            "password"
        ],
        "requiredActions": [],
        "notBefore": 0,
        "access": {
            "manageGroupMembership": true,
            "view": true,
            "mapRoles": true,
            "impersonate": true,
            "manage": true
        }
    },
    ...
]
```


GET THE GROUPS (AND SUBGROUPS) FOR DEMO USER

```
curl --location --request GET '<realm-endpoint>/users/<user-id>/groups' \
--header 'Authorization: Bearer <access_token>'
```

Response:


```
[
    {
        "id": "<group-id>",
        "name": "Demo Group",
        "path": "/Demo Group"
    },
    {
        "id": "<group-id>",
        "name": "Demo Subgroup",
        "path": "/Demo Group/Demo Subgroup"
    }
]
```

GET THE REALM ROLES FOR DEMO USER

```
curl                --location                --request                GET
'https://auth.fit.fraunhofer.de/kc/admin/realms/bimerr/users/<user-
id>/role-mappings/realm' \
--header 'Authorization: Bearer <access_token>'
```

Response:

```
[
    {
        "id": "<role-id>",
        "name": "Demo Role",
        "description": "This role is for documentations only",
        "composite": false,
        "clientRole": false,
        "containerId": "bimerr"
    }
]
```

GET LIST OF GROUPS

```
curl                --location                --request                GET
'https://auth.fit.fraunhofer.de/kc/admin/realms/bimerr/groups/' \
--header 'Authorization: Bearer <access_token>'
```

Response

```
[
    {
        "id": "<group-id>",
        "name": "Demo Group",
        "path": "/Demo Group",
        "subGroups": [
            {
                "id": "<subgroup-id>
                "name": "Demo Subgroup",
                "path": "/Demo Group/Demo Subgroup",
                "subGroups": []
            }
        ]
    },
    ...
]
```

GET DEMO GROUP'S ATTRIBUTES AND SUBGROUPS

```
curl --location --request GET '<realm-endpoint>/groups/<group-id>' \
--header 'Authorization: Bearer <access_token>'
```

Response

```
{
    "id": "<group-id>",
    "name": "Demo Group",
    "path": "/Demo Group",
    "attributes": {
        "country_code": [
            "DE"
        ],
        "description": [
            "This project is created for documentation only. It serves no
other purpose."
        ],
        "address": [
            "In the Cloud"
        ]
    },
    "realmRoles": [],
    "clientRoles": {},
    "subGroups": [
        {
            "id": "<subgroup-id>",
            "name": "Demo Subgroup",
            "path": "/Demo Group/Demo Subgroup",
            "attributes": {
                "zone_id": [
                    "1234"
                ]
            },
            "realmRoles": [],
            "clientRoles": {},
            "subGroups": []
        }
    ],
    "access": {
        "view": true,
        "manage": true,
        "manageMembership": true
    }
}
```

GET ALL REGISTRATION EVENTS

```
curl --location --request GET '<realm-
endpoint>/events?type=REGISTER&dateFrom=2020-11-01T14:30Z' \
--header 'Authorization: Bearer <access_token>'
```

Response


```
[
    {
        "time": 1604309682588,
        "type": "REGISTER",
        "realmId": "bimerr",
        "clientId": "account",
        "userId": "<user-id>",
        "ipAddress": "172.18.0.2",
        "details": {
            "auth_method": "openid-connect",
            "auth_type": "code",
            "register_method": "form",
            "redirect_uri": "<realm-endpoint>/account/login-redirect",
            "code_id": "<code-id>",
            "email": "user2@example.com",
            "username": "user2@example.com"
        }
    },
    {
        "time": 1604309677406,
        "type": "REGISTER",
        "realmId": "bimerr",
        "clientId": "account",
        "userId": "<user-id>",
        "ipAddress": "172.18.0.2",
        "details": {
            "auth_method": "openid-connect",
            "auth_type": "code",
            "register_method": "form",
            "redirect_uri": "<realm-endpoint>/account/login-redirect",
            "code_id": "<code-id>",
            "email": "user1@example.com",
            "username": "user1@example.com"
        }
    }
]
```


Look into Keycloak docs (Under Resources/Realm Admin, Find GET /{realm}/events) for other query arguments.

GET ROLE AND GROUP CHANGE EVENTS

```
curl      --location      --request      GET      '<realm-endpoint>/admin-
events?resourceTypes=GROUP_MEMBERSHIP&resourceTypes=REALM_ROLE_MAPPING&date
From=2020-11-05T14:30Z' \
--header 'Authorization: Bearer <access_token>'
```

Response


```
[
    {
```

```
        "time": 1604589885000,
        "realmId": "bimerr",
        "authDetails": {
            "realmId": "master",
            "clientId": "<client-id>",
            "userId": "<user-id>",
            "ipAddress": "172.18.0.2"
        },
        "operationType": "CREATE",
        "resourceType": "GROUP_MEMBERSHIP",
        "resourcePath": "users/<user-id>/groups/<group-id>"
    },
    {
        "time": 1604589202000,
        "realmId": "bimerr",
        "authDetails": {
            "realmId": "master",
            "clientId": "<client-id>",
            "userId": "<user-id>",
            "ipAddress": "172.18.0.2"
        },
        "operationType": "CREATE",
        "resourceType": "REALM_ROLE_MAPPING",
        "resourcePath": "users/<user-id>/role-mappings/realm"
    }
]
```


Look into Keycloak docs (Under Resources/Realm Admin, Find GET /{realm}/admin-events) for other query arguments.

CLIENT CONFIGURATION

A client needs the following configurations to be able to access the Keycloak Admin API using an access_token obtained with Client Credentials flow:

Access type: Confidential

Service Account Enabled

Service Account Roles:

- For realm-management client → view-users, view-clients

Scope "roles" must be set as default scope, or set in token requests.

ANNEX D – DATA PROCESSOR ALERT CONFIGURATION SAMPLE

Serialization Format: YAML

```yaml
## Format:
# <event-type>:
#   <bimerrProperty>:
#     <comparison-operator>: <event-specific-operand>

# Time constants
1-hour: &1h 3600
1-day: &1d 86400
1-week: &1w 604800

# Battery alert when battery level is less than or equal (lte) to value
lowBattery:
  batteryLevel:
    lte: 20

# Offline alert when offline period for property is greater than or equal (gte
) to value in seconds
offlinePeriod:
  pingTime:
    gte: *1h
  onlineState:
    gte: *1h
  temperature:
    gte: *1d
  luminacne:
    gte: *1d
  motionState:
    gte: *1d
  powerConsumption:
    gte: *1d
  powerState: # for ac controller, only one measurement is considered
    gte: *1d
  windowState:
    gte: *1w
  doorState:
    gte: *1w
```

ANNEX E – DATA PROCESSOR SAMPLE DEVICE TYPE CONFIGURATION

Serialization Format: YAML

```yaml
# Each device type is defined as one array element
## Attributes:
# * `vendor` is the device vendor/manufacturer.
# * `vendorType` is device type set by the vendor.
# * `vendorProperty` is equivalent to: fibaro type, kripis measurementID, intesis property type
# * `bimerrType` is the BIMERR device type. Enum: multiSensor, windowSensor, doorSensor, heatDetector, wallPlug, COSensor, energyMeter, acController, internetGateway, zwaveGateway
# * `bimerrProperty` is the BIMERR measurement type.
#    FIBARO HCL enum: temperature, luminance, seismicLevel, accelerationLevel, motionState, windowState, doorState, alarmState, powerConsumption, onlineState
#    Intesis ACC enum: powerState, mode, setTemperature, fanSpeed, vanePositionVertical, vanePositionHorizontal, errStatus, errCode
#    Internet Gateway enum: pingTime, noderedError, unattendedUpgradesError, sshTunnelError, fibaroTunnelError, intensisDiscoveryError
# * `dataType` is the primitive measurement data type. Enum: float, bool, string
# * `unit` is the measurement according to SenML Units Registry (https://tools.ietf.org/html/rfc8428#section-12.1) (apart from MMI).
# * `description` is the description of the sensor

-
    vendor: bimerr
    # vendorType:
    # vendorProperty:
    bimerrType: internetGateway
    bimerrProperty: pingTime
    dataType: float
    # unit:
    description: Ping time from Internet Gateway to 8.8.8.8
-
    vendor: fibaro
    # vendorType:
    vendorProperty: com.fibaro.zwavePrimaryController
    bimerrType: zwaveGateway
    bimerrProperty: onlineState
    dataType: bool
    # unit:
```

```
    description: Fibaro HCL state collected by Internet Gateway

    vendor: fibaro
    vendorType: com.fibaro.multilevelSensor
    vendorProperty: com.fibaro.temperatureSensor
    bimerrType: multiSensor
    bimerrProperty: temperature
    dataType: float
    unit: Cel
    description: Temperature measurement by multi/motion sensor


    vendor: intesis
    vendorType:
    vendorProperty: ONOFF
    bimerrType: acController
    bimerrProperty: powerState
    dataType: string
    # unit:
    description: AC power state (ON|OFF)

    vendor: intesis
    vendorType:
    vendorProperty: MODE
    bimerrType: acController
    bimerrProperty: mode
    dataType: string
    # unit:
    description: AC mode (HEAT|COOL|FAN|DRY|AUTO)
```

ANNEX F – DATA PROCESSOR SAMPLE PROJECT CONFIGURATION FOR DEVICES

Serialization Format: YAML

```yaml
# This file contains the project information.

# projectID is the unique ID on BIMERR Identity Provider.
# properties object lists details of gateways, rooms, as well as other device-specific attributes.
#  Each entry is associated with a gateway, an apartment and one or more rooms.
#   Each room can be associated with devices in order to add additional parameters

# Description of attributes in each properties entry:
# - gatewayType: # Either of: fibaro, intesis
#   gatewayName:
#   apartmentName:
#   ifcZoneID:
#   macAddress: # Intesis only, format: 123456ABCDEF
#   rooms:
#     - roomID: # For fibaro only; it must match fibaro's Room ID. Not used for Intesis.
#       roomName:
#       ifcSpaceID:
#       deviceOverrides: # To add additional metadata. For intesis, it is a partial source of device info.
#         - deviceID: # For fibaro, it must match fibaro's Device ID. For intesis, it must be unique across all controllers
#           deviceName: # Intesis only
#           loadType: # Either of: space, apartment, DHW, HVAC, light, other

projectID: a14ebdfb-6846-4066-9029-9d391463438a

properties:

- gatewayType: fibaro
  gatewayName: Bimerr
  apartmentName: apt1
  ifcZoneID: zone1
  rooms:
    - roomID: 219
      roomName: Living Room
      ifcSpaceID: space1
```

```
    - roomID: 220
      roomName: Bedroom 1 (master)
      ifcSpaceID: space2
      deviceOverrides:
        - deviceID: 8
          loadType: HVAC
    - roomID: 221
      roomName: Bedroom 2
      ifcSpaceID: space3
      deviceOverrides:
        - deviceID: 6
          loadType: HVAC
    - roomID: 222
      roomName: Office
      ifcSpaceID: space4
    - roomID: 223
      roomName: Kitchen
      ifcSpaceID: space5
      deviceOverrides:
        - deviceID: 4
          loadType: HVAC
    - roomID: 224
      roomName: Corridor
      ifcSpaceID: space6

- gatewayType: intesis
  gatewayName: acController-1
  apartmentName: apt1
  ifcZoneID: zone1
  macAddress: CC3F1D0214B2
  rooms:
    - roomName: Living room
      ifcSpaceID: space1
      deviceOverrides:
        - deviceID: 1
          deviceName: Living Room AC
          loadType:
```

ANNEX G – SAMPLE THING DESCRIPTION FOR A WIRELESS LUMINANCE SENSOR

Serialization Format: JSON


```
{
    "@context": [
        "https://www.w3.org/2019/wot/td/v1",
        "https://bimerr.fit.fraunhofer.de/td-context"
    ],
    "apartmentName": "phil",
    "created": "2020-09-10T16:34:02.458250535Z",
    "id": "de:fitdev:Fib-Philip:8",
    "ifcSpaceID": "n/a",
    "ifcZoneID": "n/a",
    "links": [
        {
            "href": "http://192.168.10.170/api/devices/8",
            "rel": "fibaroDeviceEndpoint"
        },
        {
            "href": "http://192.168.10.170/api/panels/event?deviceID=9",
            "propertyName": "motionState",
            "rel": "fibaroEventEndpoint"
        },
        {
            "href": "http://192.168.10.170/api/panels/event?deviceID=10",
            "propertyName": "temperature",
            "rel": "fibaroEventEndpoint"
        },
        {
            "href": "http://192.168.10.170/api/panels/event?deviceID=11",
            "propertyName": "luminance",
            "rel": "fibaroEventEndpoint"
        },
        {
            "href": "http://192.168.10.170/api/panels/event?deviceID=12",
            "propertyName": "seismicLevel",
            "rel": "fibaroEventEndpoint"
        }
    ],
    "meta": {
        "fibaro:created": "2021-01-22T09:14:15.000Z",
        "fibaro:deviceBaseType": "com.fibaro.device",
        "fibaro:deviceID": 8,
        "fibaro:deviceType": "com.fibaro.zwaveDevice",
        "fibaro:gatewayName": "Fib-Philip",
        "fibaro:modified": "2021-01-22T09:14:15.000Z",
        "fibaro:roomID": 224,
        "fibaro:roomName": "Living room",
        "fibaro:sectionID": 221,
        "fibaro:sectionName": "First floor"
    },
    "modified": "2021-02-08T19:17:04.293101009Z",
    "projectID": "c64a1750-8da6-4cd7-a9d5-dd0437ae3cf6",
    "properties": {
        "batteryLevel": {
            "description": "Device battery level in percentage",
            "forms": [
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href":  "https://bimerr.fit.fraunhofer.de/historical-datastore/data/de:fitdev:Fib-
Philip:8:batteryLevel",
                    "op": [
                        "readproperty"
                    ]
                },
```

```
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href": "http://172.17.0.1:8085/data/de:fitdev:Fib-Philip:8:batteryLevel",
                    "op": [
                        "readproperty",
                        "writeproperty"
                    ],
                    "security": "local_access"
                }
            ],
            "unit": "%EL"
        },
        "luminance": {
            "description": "Light intensity measurement by multi/motion sensor",
            "forms": [
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href":  "https://bimerr.fit.fraunhofer.de/historical-datastore/data/de:fitdev:Fib-
Philip:8:luminance",
                    "op": [
                        "readproperty"
                    ]
                },
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href": "http://172.17.0.1:8085/data/de:fitdev:Fib-Philip:8:luminance",
                    "op": [
                        "readproperty",
                        "writeproperty"
                    ],
                    "security": "local_access"
                }
            ],
            "unit": "lx"
        },
        "motionState": {
            "description": "Motion state (true|false) by multi/motion sensor",
            "forms": [
                {
                    "contentType": "application/senml+json; dataType=bool",
                    "href":  "https://bimerr.fit.fraunhofer.de/historical-datastore/data/de:fitdev:Fib-
Philip:8:motionState",
                    "op": [
                        "readproperty"
                    ]
                },
                {
                    "contentType": "application/senml+json; dataType=bool",
                    "href": "http://172.17.0.1:8085/data/de:fitdev:Fib-Philip:8:motionState",
                    "op": [
                        "readproperty",
                        "writeproperty"
                    ],
                    "security": "local_access"
                }
            ]
        },
        "seismicLevel": {
            "description": "Seismic intensity in modified Mercalli intensity scale (MMI) by multi/motion
sensor",
            "forms": [
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href":  "https://bimerr.fit.fraunhofer.de/historical-datastore/data/de:fitdev:Fib-
Philip:8:seismicLevel",
                    "op": [
                        "readproperty"
                    ]
                },
```

```
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href": "http://172.17.0.1:8085/data/de:fitdev:Fib-Philip:8:seismicLevel",
                    "op": [
                        "readproperty",
                        "writeproperty"
                    ],
                    "security": "local_access"
                }
            ],
            "unit": "MMI"
        },
        "temperature": {
            "description": "Temperature measurement by multi/motion sensor",
            "forms": [
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href":  "https://bimerr.fit.fraunhofer.de/historical-datastore/data/de:fitdev:Fib-
Philip:8:temperature",
                    "op": [
                        "readproperty"
                    ]
                },
                {
                    "contentType": "application/senml+json; dataType=float",
                    "href": "http://172.17.0.1:8085/data/de:fitdev:Fib-Philip:8:temperature",
                    "op": [
                        "readproperty",
                        "writeproperty"
                    ],
                    "security": "local_access"
                }
            ],
            "unit": "Cel"
        }
    },
    "roomName": "Living room",
    "security": "bearer_sc",
    "securityDefinitions": {
        "basic_sc": {
            "in": "header",
            "scheme": "basic"
        },
        "bearer_sc": {
            "in": "header",
            "scheme": "bearer"
        },
        "local_access": {
            "description": "Binding of server to localhost or docker network interfaces",
            "scheme": "nosec"
        }
    },
    "site": "fitdev",
    "title": "Motion Sensor",
    "type": "multiSensor"
}
```

ANNEX H – SAMPLE DEVICE ALERT EMAIL

Alert Type	Device Title	Device/Property	Last Measurement	Timestamp
offlinePeriod	Kitchen Door	gr:conkat:Bimerr:54:temperature	18.8	2021-01-20T11:15:35.000Z
offlinePeriod	Wall Plug	gr:conkat:Bimerr:6:powerConsumption	0	2021-01-20T11:20:00.000Z
offlinePeriod	Wall Plug	gr:conkat:Bimerr:8:powerConsumption	4.9	2021-01-20T11:20:00.000Z
offlinePeriod	Wall Plug	gr:conkat:Bimerr:4:powerConsumption	0.6	2021-01-20T11:20:00.000Z

You have received this email because of your Device Maintainer role for conkat project in BIMERR. To opt out, please send an email to bimerr-alerts-opt-out@fit.fraunhofer.de.