HORIZON-CL5-2022-D4-02

EUROPEAN COMMISSION

European Climate, Infrastructure and Environment Executive Agency

Grant agreement no. 101123238

VELIXIA

BUILDINGS AS ACTIVE UTILITY NODES

Smart Grid-Efficient Interactive Buildings

Deliverable D3.1

Platform External Communication and Common Information Management

Project acronym	EVELIXIA
Full title	Smart Grid-Efficient Interactive Buildings
Grant agreement number	101123238
Topic identifier	HORIZON-CL5-2022-D4-02-04
Call	HORIZON-CL5-2022-D4-02
Funding scheme	HORIZON Innovation Actions
Project duration	48 months (1 October 2023 – 30 September 2027)
Coordinator	ETHNIKO KENTRO EREVNAS KAI TECHNOLOGIKIS ANAPTYXIS (CERTH)
Consortium partners	CERTH, RINA-C, CEA, CIRCE, UBE, HAEE, IESRD, UNIGE, SOLVUS, R2M, EI-JKU, FHB, EEE, EG, ÖE, PINK, TUCN, DEER, TN, ENTECH, SDEF, EGC, KB, AF, Sustain, NEOGRID, MPODOSAKEIO, DHCP, HEDNO, BER, MEISA, ITG, NTTDATA, TUAS, NEOY, HES-SO
Website	https://www.evelixia-project.eu/
Cordis	https://cordis.europa.eu/project/id/101123238

Disclaimer

Copyright Message

ACKNOWLEDGMENT

This project has received funding from the European Union's Horizon Europe Framework Programme for Research and Innovation under grant agreement no 101123238. ***Disclaimer:*** *The European Commission is not responsible for any use made of the information contained herein. The content does not necessarily reflect the opinion of the European Commission.*

Deliverable D3.1

Platform external communication and common information management

Deliverable number	D3.1
Deliverable name	EVELIXIA platform external communication and common information management
Lead beneficiary	CIRCE
Description	This deliverable is directly linked to the activities foreseen in Task 3.1 and Task 3.2, consolidating all foreseen technical developments on EVELIXIA's platform interoperability, appropriate connectors, common information modelling, and knowledge repository. This report is considered as the first version of D3.2.
WP	WP3
Related task(s)	T3.1, T3.2
Type	Report
Dissemination level	Public
Delivery date	14.04.2025
Main authors	Alberto Moreno (CIRCE)
Contributors	

Document history

Version	Date	Changes	Author
V1 – first draft shared with contributing partners and reviewers	28.01.2025		CIRCE
1st review	01.02.2025		HEDNO
V1 – consolidated version	20.02.2025		CIRCE
2nd review	01.02.2025		NEOGRID
V2 – consolidated version	20.03.2025		CIRCE
Final version	31.03.2025		CIRCE
Final deliverable submission	14.04.2025		CERTH

ABBREVIATIONS

Abbreviation	Name
API	(Application Programming Interface): A set of protocols and tools for building software and applications, allowing different systems to communicate with each other.
CIM	(Common Information Model): A standard for representing data and information in a consistent and interoperable manner, often using ontologies like SAREF.
NLP	(Natural Language Processing): A field of artificial intelligence that focuses on the interaction between computers and humans through natural language.
Pub/Sub	(Publish/Subscribe): A messaging pattern where senders (publishers) send messages to a topic, and receivers (subscribers) receive messages from that topic.
SAREF	(Smart Appliances REFerence ontology): An ontology developed to enable interoperability between smart appliances and other devices.
SPARQL	(Protocol and RDF Query Language): A query language used to retrieve and manipulate data stored in Resource Description Framework (RDF) format.

GLOSSARY OF TERMS

Abbreviation	Name
Authentication	The process of verifying the identity of a user or system.
Authorization	The process of granting or denying access to resources based on the authenticated identity.
Data Broker	A middleware component that manages the distribution and routing of data between different parts of the system, often using pub/sub mechanisms.
Deployment Pipeline	A series of automated processes that manage the deployment of software from development to production environments.
Field Layer	The layer in an architecture that includes devices and sensors collecting data from the physical environment.
Knowledge Graph:	A structured representation of data that enables semantic queries and insights, often using technologies like SPARQL.
Northbound Open API Connector	A module that facilitates communication between the data management layer and the application/services layer.
Ontology	A formal representation of knowledge as a set of concepts and the relationships between those concepts.
Security Measures	Techniques and practices implemented to protect data and systems from unauthorized access and threats.
Semantic Query	A query that uses semantic technologies to retrieve data based on the meaning and relationships of the data.
Service Broker	A component that manages service requests and responses between different parts of the system.
Southbound Open API Connector	A module that facilitates communication between the data management layer and field layer devices.

TABLE OF CONTENTS

LIST OF FIGURES

EXECUTIVE SUMMARY

This document provides a comprehensive overview of the development processes for the data management layer, detailing the design, implementation, and deployment of its key software components, to efficiently collect, process, and expose data from pilot sites.

The development process involves configuring communication protocols to ensure seamless data transmission between data management layer (WP3) and the applications/services layer (WP4). Additionally, the process involves designing workflows for periodical data ingestion and routing, ensuring data quality and consistency through comprehensive validation and error handling mechanisms and secured data access and transmission.

Furthermore, the data management layer leverages advanced semantic querying and data integration capabilities to manage pilot sites data in a structured and meaningful way, facilitating the extraction of insights and relationships and so adding context and knowledge to information.

By following these development processes, the Data management layer is constructed to meet the requirements of collecting, processing, and exposing pilot sites data, ensuring data integrity, security, and accessibility through the EVELIXIA's architecture.

This document outlines the communication and data management processes of EVELIXIA that will support the delivery of the platform's first integrated version by M25, within the scope of Task 5.1. A final, updated version will be delivered as D3.2 in M33.

1 INTRODUCTION AND OBJECTIVES

This chapter defines the objectives, scope, context, and structure of the deliverable. It also outlines its relationship with other tasks within the project. To this end D3.1 delivers the EVELIXIA platform external communication and common information management, according to the EVELIXIA's interoperability and BRIDGE alignments (T3.1) and the Common Information modelling and context knowledge repository (T3.2).

1.1 Scope and Context

This deliverable is directly linked to the activities foreseen in Task 3.1 and Task 3.2, consolidating all foreseen technical developments on EVELIXIA's platform interoperability, appropriate connectors, common information modelling, and knowledge repository.

The scope of this document, titled "EVELIXIA platform External Communication and Common Information Management," encompasses the technical developments and deployments within the data management layer of the EVELIXIA platform. This layer serves as a middleware, facilitating seamless data integration and communication between the field layer and the application/services layer.
The document aims to provide a comprehensive overview of the components developed to enhance EVELIXIA platform interoperability, including connectors, data routing mechanisms, semantic querying capabilities, and standardized information modelling.

The development concentrates on the design, implementation, and individual functionalities of each component within the pilot sites data management system:

- Detailed descriptions of each component (Southbound Open API Connector, Data Broker, Northbound Open API Connector, Knowledge Graph).
- Development processes, including requirement analysis, design, implementation, and testing for each component.
- Dependencies and deployment strategies specific to each component.

1.2 Interaction with other Tasks and Work Packages

Deliverable D3.1 concisely presents the technical developments and deployments within the data management layer, based on input from:

- ✓ WP1 (D1.3 & D1.7)

The software components developed in tasks T3.1 and T3.2 are designed to interact closely with and depend on the developments from:

- ✓ WP2 (T2.4)
- ✓ WP3 (T3.3, T3.4&T3.5)
- ✓ WP4 (T4.6)
- ✓ WP5 (T5.2)

1.3 Structure of this deliverable

The deliverable is constituted by the following chapters which are interrelated and provide an overall analysis of the EVELIXIA platform architecture.

- Chapter Introduction and Objectives
- Chapter Software Deployment Strategy: Strategy, environment setup, and deployment pipeline.
- Chapter Software Components Development: Details on key components like the Southbound and Northbound Open API Connectors, Data Broker, Knowledge Graph, and Common Information Model (CIM).
- Chapter Conclusion: Summary and future enhancements.
- Chapter References

2 SOFTWARE DEVELOPMENT AND DEPLOYMENT STRATEGY

The development strategy for the innovative project on smart grid efficient interactive buildings involves a multi-faceted approach to ensure seamless integration and efficient operation of various software components.

The data broker, implemented using Apache Airflow, orchestrates and manages data workflows, ensuring timely and reliable data processing. The southbound open API connector integrates data from pilot sites, providing a unified data stream for further processing. The northbound open API connector interacts with the service broker's API, facilitating communication and data exchange between different system components. The knowledge graph, utilizing a GraphDB to store the SAREF ontology as a Common Information Model, enables advanced semantic queries. These queries are defined based on business language questions regarding the common operations of the pilot sites, ensuring that the system can provide meaningful insights and support decision-making processes.

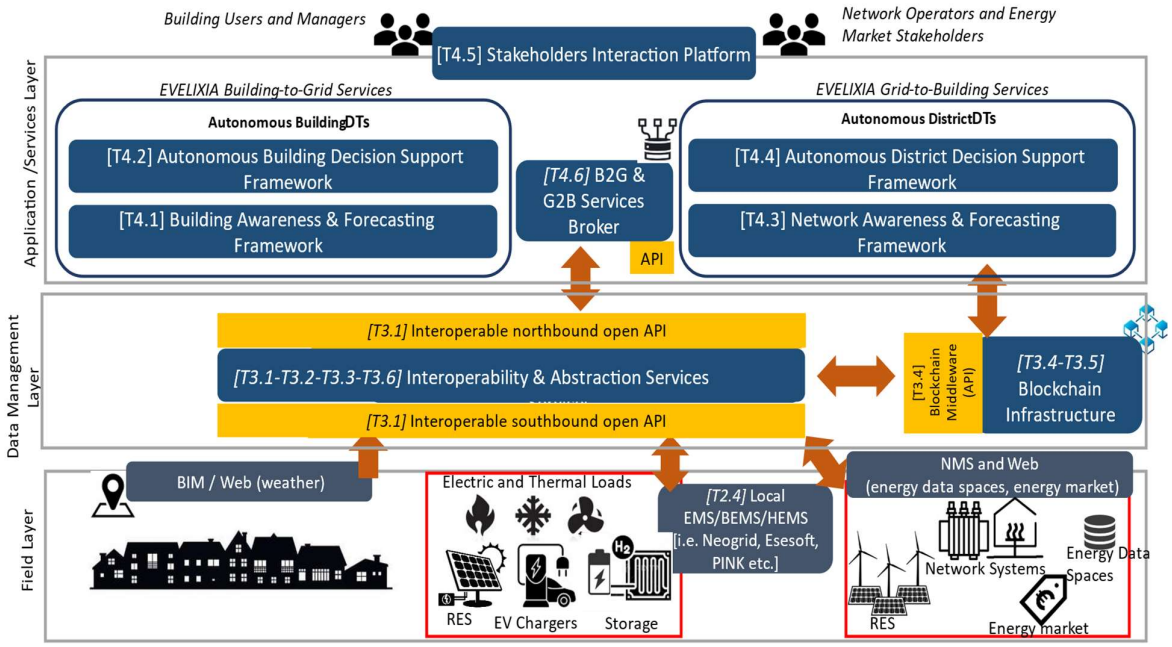

Figure 1: – Overall EVELIXIA Conceptual Architecture

2.1 Data Management Layer deployment approach

The data management layer serves as a critical middleware component within the EVELIXIA platform, facilitating seamless data integration and communication between the field layer and the application/services layer.

This layer is designed to handle diverse data sources, ensuring robust data acquisition, processing, and distribution. Key components include the Southbound and Northbound Open API Connectors, which enable interaction with field devices and application/services layers, respectively; the Data Broker, which manages data routing and pub/sub mechanisms; the Knowledge Graph, which provides semantic querying capabilities for deeper insights; and the Common Information Model (CIM), which standardizes data exchange using the SAREF ontology.

The purpose of **T3.1** developments is to have a standard pipeline to read data from certain PS (Greek, Austrian, Danish) and send RAW data to T3.3 modules and store healed data.
The purpose of **T3.2** developments is to have a small list of (about five) general business queries which describe some assets data/metadata and their relationships (context) to be translated into semantic queries.

Together, these components make up the data management layer, also known as middleware layer, enhancing the interoperability, security, and scalability of the EVELIXIA platform.

The development process for each component involves requirement analysis, design, implementation, and testing. The deployment process includes containerization, deployment to cloud or on-premises servers, and ongoing monitoring and maintenance. Dependencies for each component are carefully managed to ensure seamless integration and operation.

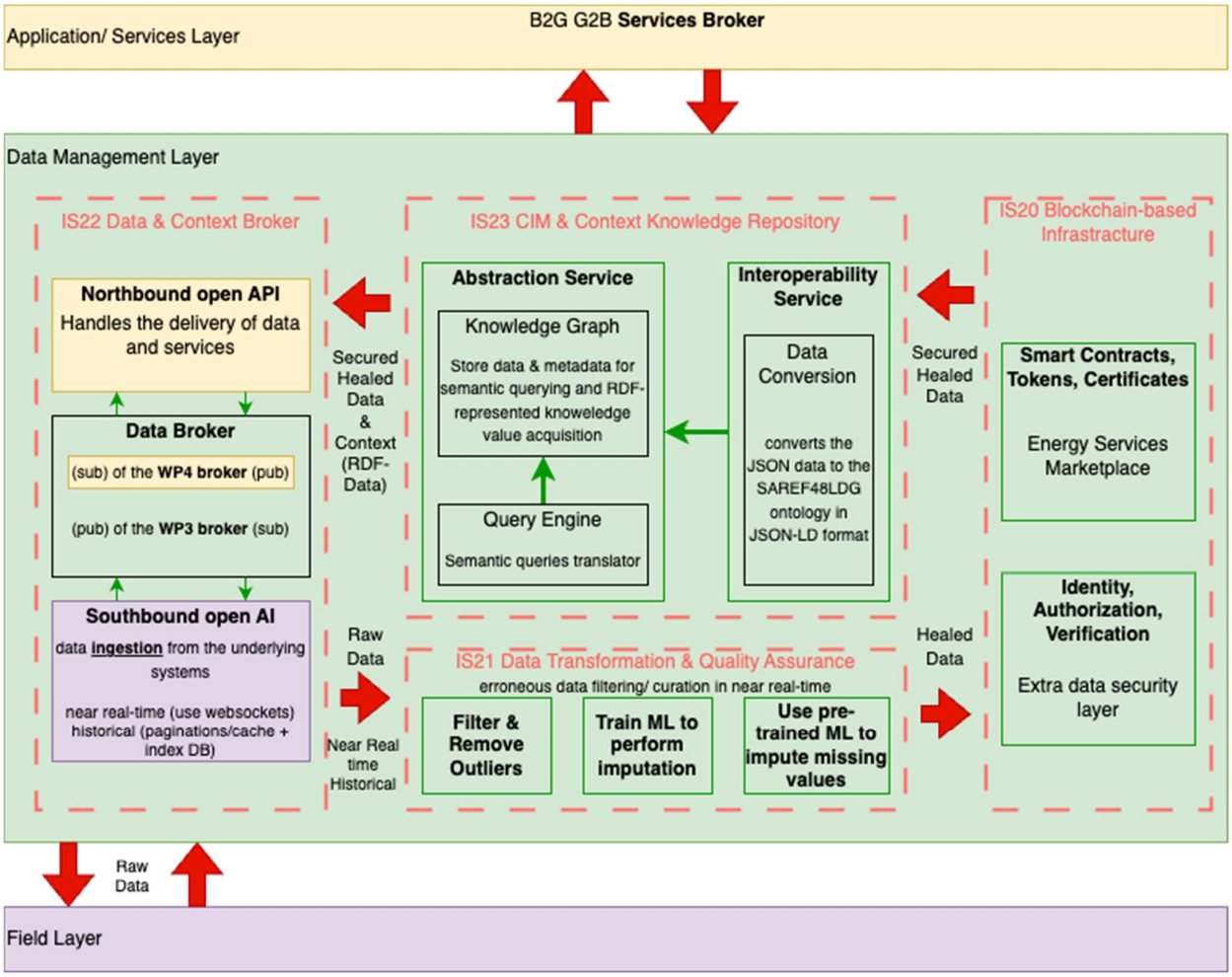

Figure 2: Data Management High Level Architecture

2.2 Tools and technologies used

1. **Programming Languages**:
   - Python for developing RESTful APIs (southbound) and data gathering scripts (in Apache workflow).
   - Node.js for developing Southbound Open API connector.
2. **Frameworks**:
   - Apache Airflow for periodical data gathering and workflow orchestration.

3. **Databases**:
    - PostgreSQL for airflow pilot site data and metadata as intermediate data storage.
    - GraphDB as the knowledge graph database managing the SAREF ontology to execute SPARQL queries
4. **Containerization and Orchestration**:
    - Docker for containerizing applications (southbound API and PostgreSQL)
5. **Visualization**
    - Grafana to visualize data gathered from pilot sites
6. **Resources**
    - Every software component has been deployed on premises in a physical server with 32 cores and 64GB of RAM.

2.3 Environment setup

For **T3.1** developments:

Two databases: one for raw data and another for healed data.

- Raw data (PostgreSQL) will contain tables that will be exposed in a REST API to be used by T3.3 modules (GET).
- Healed data for storing (POST) T3.3 curated info

Temporary endpoints were created and deactivated as soon as real API PS became available (T2.4)

In this first iteration for M18, postman collections have been used to test the endpoints.

For **T3.2** developments:

GraphDB storage for the CIM (SAREF ontology) which supports SPARQL queries through HTTP Requests.

2.4 Requirements considerations

Regarding pilot sites data:

- Temporary endpoints have been created for testing purposes and will be deactivated as soon as pilot site APIs are developed (T2.4)
- No RAW data will be sent to the service broker and so no specific Kafka topic will be mounted for this purpose.
- RAW data received by the pilot sites is stored as it is captured (JSON format)
- Each pilot site can have its own data model; the SAREF compliant conversion/wrap up will be implemented within the data broker
- The potential 'delay' in receiving healed data is not considered as a problem because data will be request during all day.
- A POSTMAN API collection with all the available APIs from pilot sites data (Greek, Austrian, Denmark) has been prepared and periodic data gathering tasks have been scheduled through the usage of Apache Airflow EVELIXIA platform.
- There will be an endpoint for RAW data (also needed for IS21 T3.3), which takes place before the semantic engine (T3.2), which will interact with WP4 developments.
- There will be two separated streams: one for reading RAW data and the other one for writing HEALED data.

Regarding data broker:

- Will be the task scheduler gathering data periodically from pilot sites API
- Will act as publisher (PUB) for the other components, which will act as subscribers (SUB).
- There will not be a bidirectional communication between data and service brokers (as no MQTT channel will be created in this first prototype)
- Service Broker (WP4) will behave as publisher (PUB) for the Data Broker, which will act as subscriber (SUB).
- Authentication/Authorization mechanism have been established between data and service broker, designed in T4.5

Regarding semantic queries:

- Common Information Model (CIM) is the already defined SAREF ontology
- The preliminary semantic queries defined in the first prototype (M18 - march '25) will be exposed through a dedicated POSTMAN collection.

Other considerations:

- Blockchain infrastructure will only oversee the energy transactions and not of the security and access control, despite serving as an extra security layer
- Authentication and Authorization:
    - Implement OAuth 2.0 for securing API endpoints.
- Data Encryption:
    - Data from pilot sites is stored unencrypted
    - Encrypt data in transit using TLS/SSL for all communications between components.
- Compliance and Auditing:
    - Ensure compliance with relevant regulations and standards (GDPR, ISO 27001).

3 SOFTWARE COMPONENTS DEVELOPMENT

This section provides a detailed overview of the key components developed within the data management layer. Each component plays a crucial role in facilitating seamless data integration and communication across the EVELIXIA platform.

3.1 Southbound Open API Connector

3.1.1 Description and functionality

Description

The Southbound Open API Connector is a critical component designed to interface with IoT field devices (sensors) to collect raw data and make it available for further processing. This connector acts as a bridge between the physical devices and the data management layer, ensuring that sensor data is periodically captured and made accessible through a standardized API.

Functionality:

1. **Data Collection**:
   - Periodically fetches raw data from Pilot sites data using HTTPS as the communication protocol
   - Supports multiple sensor types and data formats to ensure comprehensive data collection.

2. **Data Aggregation**:
   - Aggregates data from multiple sensors to provide a unified view.
   - Ensures data consistency and accuracy during the aggregation process.

3. **API Exposure**:
   - Provides RESTful API endpoints to expose the collected raw data.
   - Supports JSON data format to ensure interoperability and compatibility with different systems and applications.

4. **Scheduling and Automation**:
   - Implements task scheduling to automate periodic data collection using tools like Celery.
   - Ensures timely and reliable data capture from sensors.

- Provides metrics for monitoring sensor data access and usage patterns.

3.1.2 Development process

1. **Requirement Analysis**:
   - Analyze the pilot sites infrastructure (D1.3 “Pilot Site Surveys results, Use Cases definition and market needs analysis”) to define the data sources, communication protocols, and data formats of the pilot sites.
   - Determine the frequency of data collection and the aggregation logic.
   - Analyze the commercial platforms already in use for particular pilot sites
   - Prioritize according to matureness. Finally selected Austrian, Danish and Greek for the first protype (M18).

2. **API Design**:
   - Design the API endpoints for data retrieval and aggregation, as soon as new assets data is available in the pilot sites

3. **Implementation**:
   - Set up the development environment with the chosen programming language (Python and Node.js).
   - Develop the API endpoints to fetch and aggregate data from IoT sensors.
   - Implement periodic data collection using a task scheduler (Apache airflow).

4. **Testing**:
   - Perform integration testing to verify that data is correctly aggregated and served.
   - Validate the performance and reliability of the data collection process.

3.1.3 Deployment process

1. **Containerization**:
   - Create a Dockerfile to containerize the application.
   - Build the Docker image to ensure consistency across different environments.

2. **Deployment**:
   - Deploy the Docker container to a on-premises server.

3.1.4 Dependencies

- **Programming Language**: Python, Node.js
- **Task Scheduler**: Apache Airflow
- **Containerization**: Docker
- **Cloud Services**: on premises
- **Database**: PostgreSQL

3.2 Northbound Open API Connector

3.2.1 Description and functionality

Description
The Northbound Open API Connector is a crucial component designed to expose healed data to a service broker. This connector acts as an interface between the data management layer and the service broker, ensuring that the data provided is clean, validated, and ready for consumption by various services and applications.

Functionality:

1. **Data Retrieval**:
   - Fetches raw and transformed data from the Data Broker.
   - Applies data transformation and healing rules to clean and validate the data.

2. **Data Exposure**:
   - Provides RESTful API endpoints to expose the healed data to the service broker (WP4) through its API (T4.6).
   - Supports JSON data format to ensure interoperability and compatibility with the service broker.

3. **Data Security**:
   - Bypasses the authentication and authorization mechanisms (define in T4.5) to secure data access.
   - Ensures data integrity and confidentiality during transmission.

3.2.2 Development process

1. **Requirement Analysis**:
   - Define the data requirements and endpoints for the service broker.
   - Determine the data transformation and healing rules.

2. **API Design**:
   - Design the API endpoints to expose healed data.

3. **Implementation**:
    - Set up the development environment with the chosen programming language (Python with Node.js).
    - Develop the API endpoints to fetch, transform, and expose data.

4. **Testing**:
    - Perform integration testing to verify that data is correctly transformed and exposed.
    - Validate the performance and security of the API.

3.2.3 Deployment process

1. **Containerization**:
    - Create a Dockerfile to containerize the application.
    - Build the Docker image to ensure consistency across different environments.

2. **Deployment**:
    - Deploy the Docker container to an on-premises server.

3.2.4 Dependencies

- **Programming Language**: Python, Node.js
- **Task Scheduler**: Apache Airflow
- **Containerization**: Docker
- **Cloud Services**: on premises
- **Database**: PostgreSQL

3.3 Data broker

3.3.1 Description and functionality

Description
Apache Airflow serves as the Data Broker in this architecture, orchestrating the periodic capture of data from Pilot sites data and managing the data flow within the system. It ensures that data is ingested, transformed, and routed efficiently to the Northbound Open API Connector for further processing and exposure.

Functionality:
1. **Workflow Orchestration**:
    - Handles task dependencies and ensures reliable execution of data pipelines.

2. **Data Ingestion**:
   - Periodically fetches raw data from the Southbound Open API Connector.

3. **Data Transformation**:
   - Transformations on RAW data, including data healing and imputation techniques are performed in T3.3 modules, to ensure data quality.

4. **Data Routing**:
   - Routes transformed data to the Northbound Open API Connector.

3.3.2 Development process

1. **Requirement Analysis**:
   - Define the data sources, transformation rules, and routing logic.
   - Determine the scheduling frequency and workflow dependencies.
2. **Implementation**:
   - Set up the development environment with Apache Airflow.
   - Develop Python scripts to define data gathering tasks.
   - Implement data ingestion, transformation, and routing logic within the tasks.
3. **Testing**:
   - Perform integration testing to ensure workflows execute correctly.
   - Validate data quality and transformation results.

3.3.3 Deployment process

1. **Containerization**:
   - Create a Dockerfile to containerize the Apache Airflow setup.
   - Build the Docker image for the Airflow environment.
2. **Deployment**:
   - Deploy the Docker container to a cloud service provider (e.g., AWS, Azure) or on-premises server.
   - Set up a container orchestration tool (e.g., Kubernetes) if needed.
   - Configure Airflow to use a backend database and message broker (PostgreSQL).

3.3.4 Dependencies

- **Programming Language**: Python, Node.js
- **Task Scheduler**: Apache Airflow
- **Containerization**: Docker
- **Cloud Services**: on premises
- **Database**: PostgreSQL

3.4 Knowledge graph and Common Information Model (CIM)

3.4.1 Description and functionality

Description
The Knowledge Graph and Common Information Model (CIM) component leverages GraphDB to store and manage the SAREF ontology, providing a semantic layer for Pilot Sites data. This component enables advanced querying and data integration, allowing users to derive insights and relationships from the data using SPARQL queries.

Functionality:

1. **Ontology Management**:
   - Loads and manages the SAREF ontology within GraphDB.
   - Ensures the ontology is up-to-date and accurately represents the data model.

2. **Data Ingestion**:
   - Converts raw sensor data into RDF format compatible with the SAREF ontology.
   - Inserts RDF data into GraphDB using HTTPS POST requests.

3. **Semantic Querying**:
   - Provides RESTful API endpoints to execute SPARQL queries and return results.
   - Supports complex queries to derive insights and relationships from the data.

4. **Data Integration**:
   - Integrates data from multiple sources to create a unified knowledge graph.
   - Ensures data consistency and integrity within the knowledge graph.

3.4.2 Development process

1. **Requirement Analysis**:
   - Analyse that SAREF ontology is suitable to be used as an ontology, meeting the objectives of the project.
   - Determine the types of queries and data retrieval requirements.
   - Select the SAREF ontology as the Common Information Model (CIM).
2. **Ontology Setup**:
   - Download and configure the SAREF ontology.
   - Load the ontology into GraphDB using the GraphDB Workbench.
3. **Data Ingestion**:
   - Develop scripts to convert raw sensor data into RDF format.
   - Implement HTTPS POST requests to insert RDF data into GraphDB.
4. **SPARQL Query API Development**:
   - Develop an API to handle SPARQL queries and return results.
   - Implement endpoints to execute SPARQL queries via HTTPS requests.
5. **Testing**:
   - Perform integration testing to ensure data is correctly stored and retrieved.
   - Validate the performance and accuracy of SPARQL queries.

3.4.3 Deployment process

1. **GraphDB Setup**:
   - Install and configure GraphDB on a server or cloud instance.
   - Create and configure the repository for the ontology.
2. **Containerization**:
   - Create a Dockerfile for the SPARQL Query API.
   - Build the Docker image to ensure consistency across different environments.
3. **Deployment**:
   - Deploy GraphDB and the SPARQL Query API container to a cloud service provider (e.g., AWS, Azure) or on-premises server.
   - Set up a container orchestration tool (e.g., Kubernetes) if needed to manage the deployment.

3.4.4 Dependencies

- **Ontology**: SAREF
- **Graph Database**: GraphDB
- **Programming Language**: Python
- **Task Scheduler**: Apache Airflow
- **Containerization**: Docker
- **Cloud Services**: on premises

4 CONCLUSIONS

The development of the pilot sites data management system integrates several critical components, each with distinct roles and functionalities, to ensure efficient data collection, processing, and exposure. Insights from the key components and their roles in this middleware layer are listed below:

Southbound Open API Connector:

- Collects raw data from Pilot sites data and exposes it through standardized API endpoints.
- Ensures timely and reliable data capture with automated scheduling and monitoring.

Data Broker (Apache Airflow):

- Orchestrates workflows for periodic data ingestion, transformation, and routing.
- Manages the data flow within the system, ensuring data quality and consistency.

Northbound Open API Connector:

- Exposes healed data to a service broker, ensuring the data is clean and validated.
- Implements robust security measures to protect data integrity and confidentiality.

Knowledge Graph and Common Information Model (CIM):

- Utilizes GraphDB to store and manage the SAREF ontology, enabling semantic querying.
- Provides advanced data integration and querying capabilities to derive insights from pilot sites data.

This way, the system leverages advanced technologies and methodologies to provide a robust solution for managing pilot sites data.

The present document analyzes EVELIXIA's communication and data management processes, which will be deployed for the delivery of the first version of the integrated platform by M25, as part of Task 5.1. The final version of this document will be submitted in M33 (D3.2).

5 REFERENCES

[1] "Gaia-X," [Online]. Available: https://gaia-x.eu/.

[2] "Node JS," [Online]. Available: https://nodejs.org/en/.

[3] "IDS RAM 4.0," International Data Spaces Association, [Online]. Available: https://docs.internationaldataspaces.org/knowledge-base/ids-ram-4.0.

[4] A. R. A. E. A. E. Boris Otto, "GAIA-X and IDS," Berlin, Germany, January 2021.

[5] "docker," [Online]. Available: https://www.docker.com/.

[6] B. Frost, " "Atomic Design,," 20 October 2021. [Online]. Available: https://bradfrost.com/blog/post/atomic-web-design/ . [Accessed 24 May 2024].