Project acronym	B4B
Project full name	Brains for Building's Energy Systems
Grant No	MOOI32004
Project duration	4 year (Starting date May 1, 2021)

Deliverable 4.06

Reference Architecture for Smart Buildings

Lasitha Chamari, Pieter Pauwels, Ekaterina Petrova (Eindhoven University of Technology)
Jan-Willem Dubbeldam (Kropman), Niels De Jong (Cloud Energy Optimizer), Karthik Gunderi (Deerns)

Work package	WP4
Result	8
Lead beneficiary	Eindhoven University of Technology
Due Date	30 April 2023
Deliverable Status	Final
File name	B4B-WP4-D4.0.6-Reference_Architecture
Reviewers	Shobhit Chitkara (Spectral Energy)

This project received funding from Dutch Ministry of Economic Affairs and Climate Policy and Ministry of the Interior and Kingdom Relations under the MOOI program

SAMENVATTING

Brains for Buildings Energy Systems (B4B) is een meerjarig project met meerdere belanghebbenden dat tot doel heeft om slimme bouwmethoden te ontwikkelen om de activiteiten in gebouwen energie-efficiënter te maken, om flexibel energieverbruik mogelijk te maken, om comfort te verhogen van de ge-bruikers en om installatietechnisch onderhoud slimmer te maken. Dit wordt bereikt door snellere en ef-ficiëntere modellen en algoritmen voor Machine Learning en Artificial Intelligence te ontwikkelen. Het pro-ject is afgestemd op bestaande utiliteitsbouw.

Vijf werkpakketten in dit project zijn gewijd aan bovenstaande taken. Dit rapport is een uitkomst van het Werkpakket 4 (WP 4) van het B4B-project, met als thema "Data Integratie". Er zijn veel heterogene gegevensbronnen in gebouwen, b.v. sensorgegevens, 3D-modelgegevens, tijdschema's, gegevens van be-woners, gegevens over systeemonderhoud, gegevens van elektrische systemen, enz. Hun gegevenstypen, naamgevingsconventies en formaten zijn heel verschillend van gebouw tot gebouw, waardoor het moeilijk is om deze gegevens te gebruiken voor het ontwikkelen van gestandaardiseerde rekenmodellen en algo-ritmen. Gegevensbronnen of systemen communiceren bovendien met verschillende protocollen en wis-selen gegevens uit in verschillende formaten en standaarden. Daarom zijn deze gegevens vaak enkel beschikbaar in silo's. Ze zijn niet beschikbaar via één platform, waardoor er weinig tot geen interactie of koppeling of gezamenlijk gebruik is van meerderede dergelijke silo's. WP4 heeft tot doel methoden te ontwikkelen om deze gegevenssilo's te integreren voor het ontwikkelen van modellen en algoritmen voor machine learning en kunstmatige intelligentie, terwijl ook privacy en ethiek worden gegarandeerd bij het verzamelen, opslaan, integreren, delen, beheren of gebruiken van gegevens in slimme gebouwen.

Deze deliverable *"D.4.06 Reference Architecture for smart buildings"* rapporteert de resultaten van taak *T4.2.1: Ontwikkeling van een referentiesysteemarchitectuur en een reeks gegevensstroomprocedures* en taak *T4.3.1: Ontwikkeling en uitrollen van voorgestelde systeemarchitectuur en -aanpak voor data integratie in meerdere testomgevingen en field labs.* De studie begint met het verkennen van de state-of-the-art referentiearchitecturen en systeemarchitecturen met betrekking tot smart buildings. Vervolgens worden stakeholders geïdentificeerd om onderzoek te doen naar hun vereisten rond data integratie in smart buildings. Deze vereisten worden gedefinieerd in een reeks functionele vereisten om daarmee een referentie-architectuur te ontwerpen. De studie stelt een systeemarchitectuur voor met vijf lagen, gebaseerd op microservices. De geïdentificeerde functionele vereisten worden geïmplementeerd met behulp van deze verschillende microservices. De vijf lagen van de architectuur bestaan uit Databronnen, Data-integratie, Databeheer, Applicatie-logica en Visualisatie, in overeenstemming met state-of-the-art ontwikkeling.

De algemene idee van de voorgestelde systeemarchitectuur is om communicatie van verschillende gegevenstypen mogelijk te maken met data-gestuurde toepassingen in slimme gebouwen. De voorgestelde micro-services om deze communicatie te bereiken, zijn: i) drivers voor het verkrijgen van gegevens van gebouwautomatiseringssystemen, metadatabronnen en externe dataservices (e.g. weerdata), en nutsvoorzieningen, ii) real-time berichtendiensten, iii) databanken die geschikt zijn voor verschillende gegevenstypes, iv) Application Programming Interfaces (API) voor datacommunicatie, iv) verschillende smart building toepassingen en v) front-end applicaties die fungeren als visualisatietools. Aan het einde van het rapport worden verschillende voorbeeldimplementaties van smart building-toepassingen gepresenteerd als een demonstratie van de voorgestelde referentiearchitectuur.

SUMMARY

Brains for Buildings Energy Systems (B4B) is a multi-year, multi-stakeholder project that aims to develop smart building methods to enhance operations in buildings by reducing energy consumption, increasing energy flexibility, increasing occupant comfort, and improving installation maintenance costs. This will be achieved by developing faster and more efficient Machine Learning and Artificial Intelligence models and algorithms. The project is geared towards existing utility buildings, such as commercial and institutional buildings.

There are five work packages in this project dedicated to the above tasks. This report is an outcome of Work Package 4 (WP 4) of the B4B project. WP4 works under "Data integration" theme. There are many heterogeneous data sources in buildings, e.g., sensor data, 3D model data, time schedules, occupant data, asset maintenance data, electrical system data, etc. Their data types, naming conventions and formats differ, making it difficult to use these data for developing models and algorithms. Data sources or systems communicate using different protocols and exchange data in different formats and standards. Therefore, these data are often siloed, they are not available via a single platform, meaning there is little to no interaction between them. WP4 aims to develop methods to integrate these data silos for developing Machine Learning and Artificial Intelligence models and algorithms while guaranteeing privacy and ethics when collecting, storing, integrating, sharing, managing or utilizing data in smart buildings.

This deliverable, "*D.4.06 Reference Architecture for smart buildings,"* reports the outcomes of task *T4.2.1: Development of a reference system architecture and set of data flow procedures* and *T4.3.1: Development and roll-out of proposed reference system architecture and data integration methods in multiple test environments and field labs*. The study explores the state-of-the-art reference architectures and system architectures related to the smart building domain. Then it continues to identify the stakeholders involved in a smart building to gather the requirements of each. These requirements are then defined into functional requirements to design a reference architecture.

The study proposes a five-layer microservice-based architecture, where the identified functional requirements are implemented using various microservices. Five layers of the architecture consist of Data sources, Data integration, Data management, Application logic and the visualization layers. The overall idea of the proposed system architecture is to enable communication of various data types with data-driven applications in smart buildings. Proposed microservices to achieve this communication includes i) drivers for acquiring data from building automation systems, metadata sources and external data services such as weather and utility, ii) real-time message services, iii) databases suitable for different data types, iv) Application Programming Interfaces (API) for data communication, iv) various smart building applications and v) front-end applications that act as visualization tools. At the end of the report, several example implementations of smart building applications are presented as a demonstration of the proposed reference architecture.

TABLE OF CONTENTS

LIST OF FIGURES

1 INTRODUCTION

1.1 Smart building systems

Smart buildings are complex systems that combine many systems, including Building Automation Systems (BAS), IoT devices, user preferences, and analytics. With the rise of the Internet of Things (IoT) and cloud computing, smart buildings have become more sophisticated, with a vast array of connected devices and systems that need to communicate and exchange information. Advanced algorithms based on Machine Learning (ML) and advanced controls such as Model Predictive Control (MPC) can now optimize building operations based on building data and connectivity.

A smart building includes components from several different domains, such as building automation, information technology, communication networks and user interfaces. Some key components of a smart building include:

1. Building Automation System (BAS): A centralized system for monitoring and controlling building systems such as HVAC, lighting, and security.
2. IoT Devices: Various sensors, actuators, and other devices for monitoring and controlling building systems, such as temperature sensors, motion sensors, and lighting controllers.
3. Real-time Communication: A low-latency, high-bandwidth communication network and network infrastructure for real-time data exchange between the various components of the smart building system, such as Wi-Fi, Ethernet, and cellular networks. The real-time communication component is particularly important for ensuring that the smart building system can respond quickly to changes in the environment and building systems and make decisions based on real-time data.
4. Data Management: A system for collecting, storing, and analyzing data from the IoT devices and BAS, such as databases and data analytics platforms.
5. Metadata Management: A system for collecting, storing, and integrating metadata from multiple systems.
6. Data Integration: A set of APIs and protocols for integrating with other systems, such as building management systems, data analytics services, weather services, etc.
7. Analytics: Various rule-based and optimization-based control strategies are executed in buildings to achieve operational objectives related to energy efficiency, comfort and indoor environment.
8. User Interfaces: A set of user-friendly interfaces for monitoring and controlling the building systems, such as mobile apps, web applications, and screens.
9. Security: A comprehensive security solution for protecting the smart building system from cyber threats, such as firewalls, encryption, and access controls.

The complexity of smart buildings as a cyber-physical system stem from the large number of interdependent systems and technologies mentioned above, each with their own data, algorithms, and communication protocols, that must work together to achieve the desired outcomes, such as energy efficiency, user comfort, and safety. A well-designed system architecture can help to manage this complexity, ensuring the efficient and effective integration of necessary components, and ultimately delivering a building that operates at its best and meets the needs of all stakeholders involved.

1.2 Reference architecture for smart buildings

Given the diversity of the available building stock, system vendors, technologies, and operational objectives of buildings, developing a system architecture that suits all scenarios is not possible. Here, the architectural blueprints of the system architecture (reference architecture) can support the creation of suitable instances of systems by introducing reusable knowledge and best practices. The main purpose of a reference architecture is to provide (i) re-usability, (ii) best practices, (iii) abstraction levels, and (iv) serve a set of use cases [1]. A reference architecture is a standardized, reusable set of guidelines, tools, and practices for designing and building a specific solution or system [1]. It provides a blueprint for the solution's design, development, and deployment and is intended to help organizations achieve a consistent, high-quality outcome.

A lot of buildings are becoming data-driven and there is a need to integrate different data sets from multiple sources, train and run control algorithms based on the collected data. These applications need to be modular

and portable to increase their usability between different technologies and buildings [2], [3] and therefore modularity is important. A reference architecture therefore can aid the design and deployment of systems with such requirements [4]. A reference architecture can help reduce the cost, time, and risk associated with developing and deploying new systems in smart buildings.

A reference architecture can provide a smart building with,

1. **Clarity:** provides a clear and consistent framework for designing, deploying, and managing smart building systems[5]. This can help to ensure that the building systems are aligned with the overall goals and objectives of the building.
2. **Interoperability and data integration:** ensure that the various components and systems of the smart building can work together seamlessly and effectively. This includes integrating different building systems (such as HVAC, lighting), external services (like weather and grid) with control algorithms for smart building applications such as FDD and MPC [6].
3. **Security, privacy and ethical usage of data:** ensure that the smart building systems are secure and protected from cyber threats, such as hacking and data breaches. This includes implementing appropriate security measures, such as firewalls, encryption, and access controls, as well as ensuring that privacy concerns are addressed throughout the system architecture to handle sensitive information.
4. **Cost and efficiency:** optimize the costs[7] and efficiency of deploying the smart building systems by reducing the risk of compatibility issues and improving the overall performance and reliability of the building systems.
5. **Flexibility and adaptability:** provides an overview of the components and their interactions with each other to design flexible and adaptable solutions. Since changes and upgrades to the building systems and control algorithms are inevitable over time, this overview helps to ensure that the building systems can evolve and grow as needed, while still maintaining compatibility with the overall architecture.

2 STATE-OF-THE-ART REFERENCE ARCHITECTURES

In the early 2000s, the rise of the internet[1] and the rapid advancement of technology spurred the increasing importance of reference architectures in different domains such as IoT [8], Industrial IoT (IIoT) [1], manufacturing technology, autonomous systems, and smart buildings [4], [5], [9], [10]. These architectures were utilized to assist companies in the design and implementation of complex software systems such as enterprise resource planning (ERP) systems, service-oriented architectures (SOAs), and cloud computing. Reference architectures are extensively applied across several industries, including healthcare, finance, and smart buildings. They provide a standard framework for designing and implementing complex systems and are continuously evolving and adapting to new technologies and challenges. Four such domains and reference architectures in those domains are discussed in the following sections.

2.1 Smart buildings domain

MATRYCS[2] [11], [12] is a reference architecture designed for the building domain to enable the implementation of **advanced services using big data**. Main components of the architecture are, data sources, data governance, data processing, and analytics, as shown in Figure 1. The data sources layer implements various edge data connectors to collect data from various sources, including sensors, building management systems, and weather services. The governance layer handles the streaming services, semantic services, data storage and querying. The processing layer implements data analytics tools suitable for model development, model evaluation and other Machine Learning tasks. Finally, the analytics toolbox provides advanced services, such as predictive maintenance, energy management, and indoor environmental quality analysis, by utilizing the processed and stored data. The MATRYCS architecture is intended to provide a standardized and scalable framework for developing advanced services in the building domain, particularly for ML-based applications.

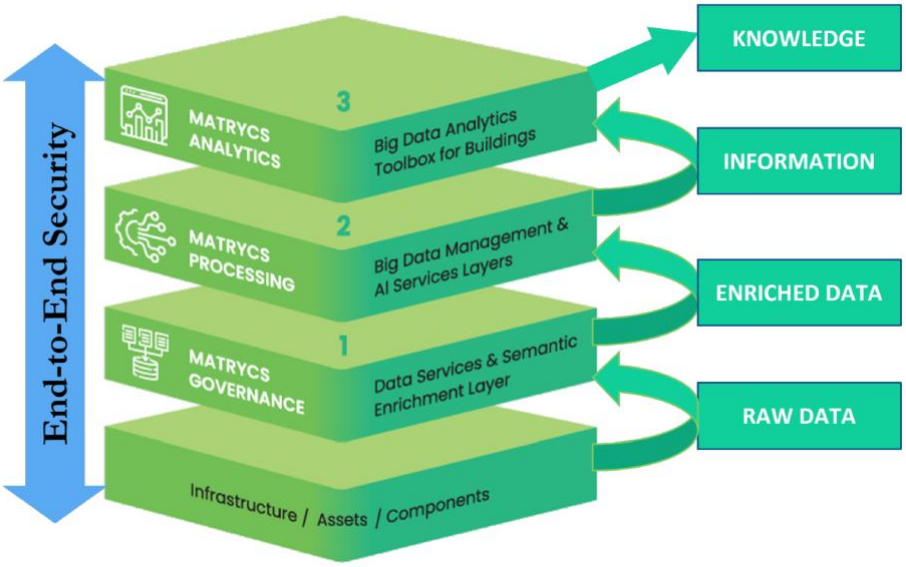

Figure 1 High level MATRYCS architecture with four layers transforming raw data to knowledge[12][12]

A detailed view of the proposed architecture and possible open source components for implementing a system is proposed by the MATRYCS project, and is shown in Figure 2. In each layer, software frameworks, databases and software libraries are proposed, which can be used in the instantiation of the reference architecture.

[1] https://www.britannica.com/technology/Internet/Foundation-of-the-Internet

[2] https://www.matrycs.eu/sites/default/files/2021-06/MATRYCS%20D2.2%20-%20Technical%20%26%20Security%20Specification_v1.0%20%282%29.pdf

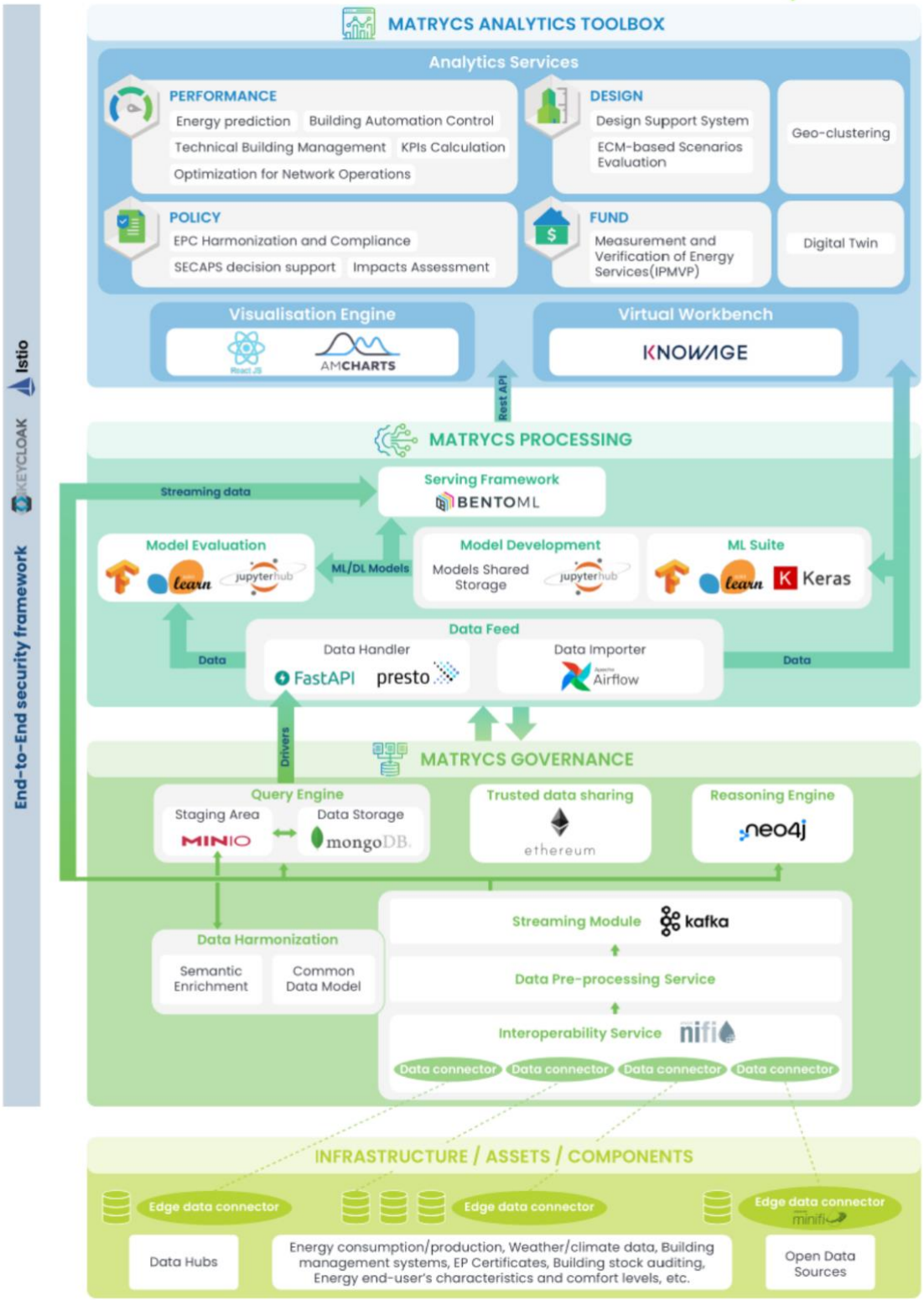

Figure 2 MATRYCS architecture and its proposed instantiation with open source technologies [13]

2.2 Manufacturing domain

The Boost4.0 Reference Architecture (RA)[3] is built with **big data in manufacturing** in mind. The architecture is composed of five horizontal layers that group various components. The integration layer facilitates the management of external data sources, infrastructure, and data ingestion. The horizontal layers are complemented by cross-cutting aspects represented in vertical components, such as communication, data sharing platforms, development, engineering, and DevOps, standards and, cybersecurity and trust. The 'Information and Core Big Data' layer consists of components responsible for data management, processing, analytics, and visualization. The Application layer implements application logic that supports specific business functionalities and exposes the functionality of lower layers through appropriate services. The Business layer forms the overall manufacturing business solution in the Boost4.0 project. The factory dimension (y-axis) indicates how physical entities are involved in the reference architecture.

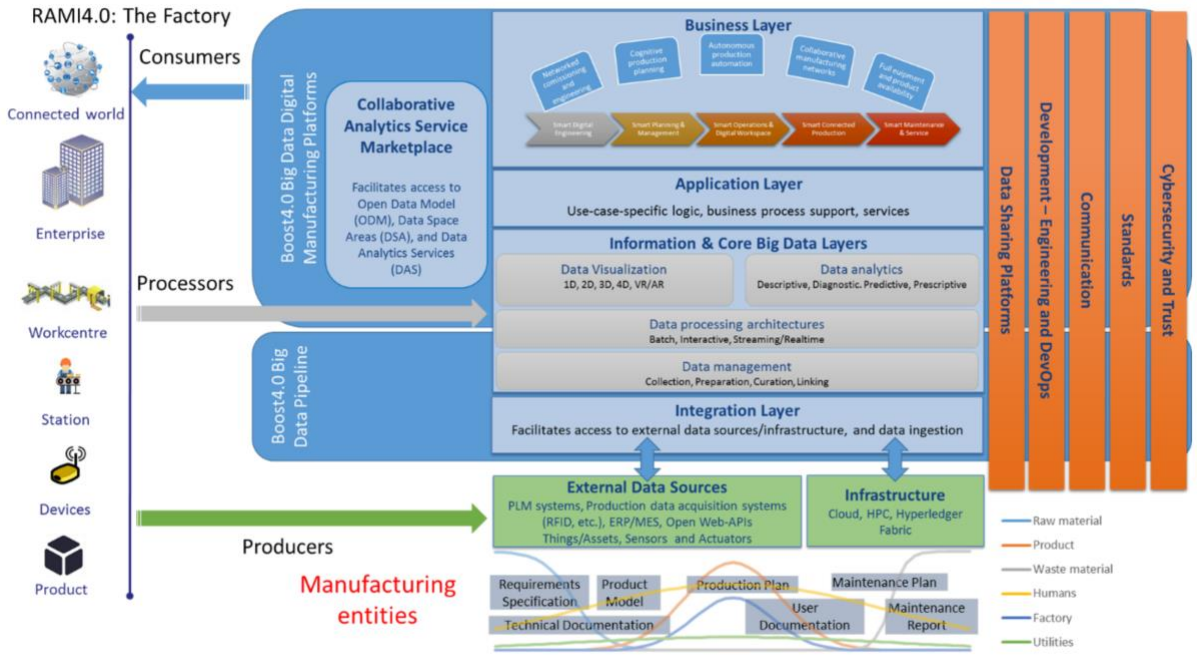

Figure 3 BOOST 4.0 Reference Architecture[14]

2.3 IIoT systems domain

The IIRA Reference architecture[4] is built with **IIoT systems** in mind. This reference architecture utilizes the ISO/IEC/IEEE 42010[5] architecture description specification[6] as its architecture framework and employs *stakeholders, their concerns and viewpoints (business view, usage view, functional view, and implementation view)* as its architecture representation to describe and analyze important common architecture concerns for IIoT systems. It focuses on architectural concerns commonly found in IIoT systems and categorizes them into viewpoints with their respective stakeholders. The IIRA models are chosen to address concerns at an appropriate level of abstraction which can be used as a starting point for concrete architectures and can be extended and enriched based on specific system requirements. System architects can use IIRA as a base framework to develop their concrete architecture.

ISO/IEC/IEEE 42010 is intended for use by software architects, stakeholders, and anyone involved in the development or evaluation of software, systems, or enterprises. It is applicable to a wide range of systems, including those in the domains of information technology, manufacturing, transportation, healthcare, and defense. The ISO/IEC/IEEE 42010 standard emphasizes the importance of considering multiple viewpoints

[3] https://boost40.eu/wp-content/uploads/2020/11/D2.5.pdf

[4] https://www.iiconsortium.org/wp-content/uploads/sites/2/2022/11/IIRA-v1.10.pdf

[5] https://www.iso.org/obp/ui/#iso:std:iso-iec-ieee:42010:ed-2:v1:en

[6] ISO/IEC/IEEE 42010 is an international standard that provides a framework and guidelines for describing the architecture of software, systems, and enterprises. It defines the terminology, concepts, and constructs that are used in architecture descriptions such as architecture, architecture description (AD), and architecture description language (ADL), architecture description framework (ADF), and provides a common language and structure for stakeholders to communicate and understand the architecture of a system.

when describing architecture and provides guidance on how to develop and use these viewpoints to analyze, evaluate, and communicate architecture. It also addresses the relationships between architecture and other system characteristics, such as requirements, design, implementation, and testing.

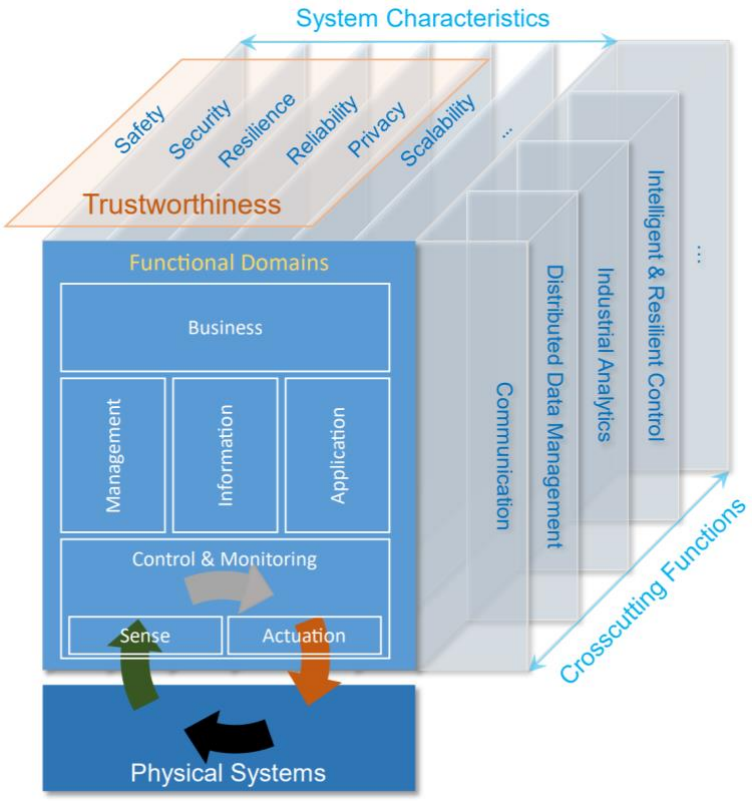

Figure 4 High-level IIRA Reference Architecture with functional domains, cross-cutting functions and system characteristics

2.4 Autonomous systems domain

Marosi et al. [1] introduces a cloud-agnostic data analytics platform for state-of-the-art autonomous systems. The platform is built from open-source reusable components and serves as a blueprint for processing and analyzing structured and unstructured data from IoT-based cases. It is based on industry's best practices and IoT-based data analytics platforms. The authors propose this platform as a baseline for future reference architectures in autonomous systems. Many components of this architecture are highly relevant to the smart building analytics domain, such as real-time data ingestion, data storage and real-time analytics. This system architecture can be considered as a low-level architecture which provides direct guidelines for implementations using available software components, which can be valuable for instantiating such reference architectures in real world applications.

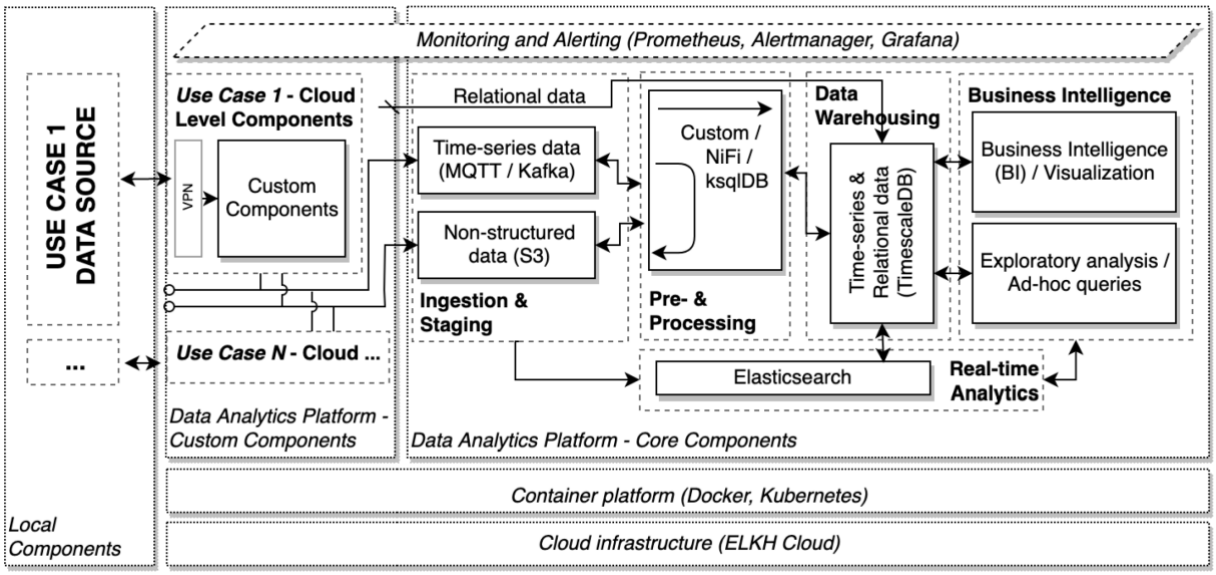

Figure 5 Data analytics platform architecture [1](image used with permission from authors)

2.5 Commercially available smart building system architectures

There are architectures (not necessarily reference architectures) designed and used by companies which are tailored for smart building requirements. These architectures are low-level and close to implementation with specifications and Application Programming Interface (API) documentation. One such commercial system architecture is Building Information System (BIS)[7]. The architecture of this system is shown in Figure 6. Here, the Building Operating System (BOS) consolidates and standardizes building data. At the core of this system is the digital twin, which serves as the unified and contextualized data model for describing the building's state, history, and behavior. The BOS itself is the software responsible for managing this digital twin database. BOS API acts as the singular and secure entry point for all building data, regardless of its source. This API provides seamless access to the digital twin and its associated data, ensuring that different building systems and applications can easily and reliably interact with each other.

[7] https://resourcecenter.en.spinalcom.com/project-resources/the-bos-spinalcore/system-architecture

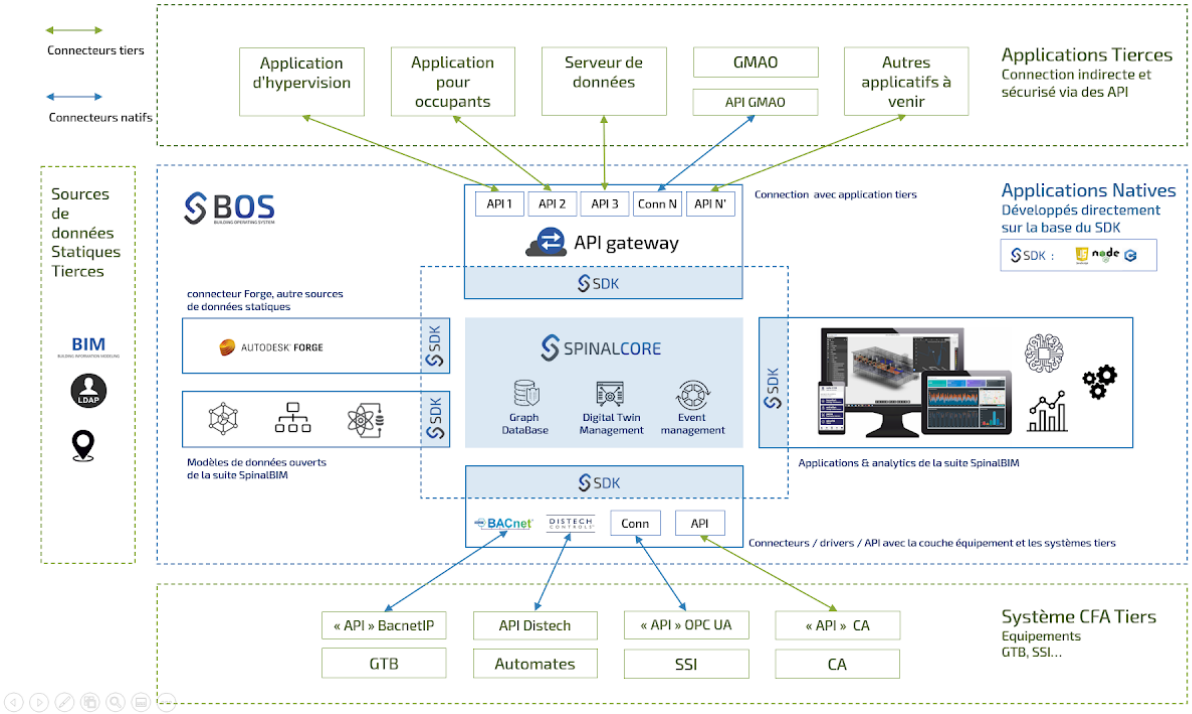

Figure 6 BIS System Architecture (image from https://resourcecenter.en.spinalcom.com/project-resources/the-bos-spinalcore/system-architecture)

The architecture in Figure 7 is a good example for a high-level architecture used by a commercial company. This architecture belongs to Cloud Energy Optimizer, a project partner in the Brains for Buildings project. The idea of this architecture is to communicate the main components in the system, which are, the building, data connectors, analytics environment and the historical database of building data.

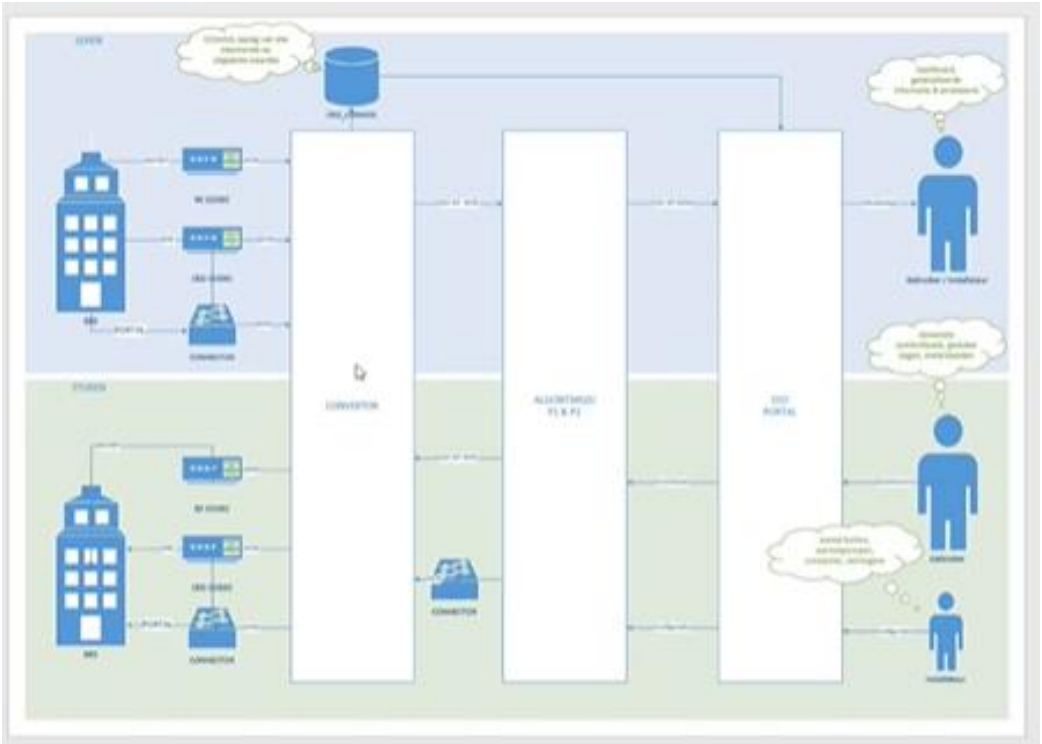

Figure 7 Cloud Energy Optimizer system architecture.

Figure 8 illustrates the system architecture of Kropman's InsiteView[8], another partner company in the Brains for Buildings project. This architecture is detailed with the protocol level interactions with each component in the system and therefore a low-level system architecture. The diagram shows the interaction and communication of data between various components such as BAS systems (Priva, Johnson Control), external serivices (weather, Electric Vehicle charging), databases and ML algorithm development environment. The InsiteReports Server provides historical data while an MQTT broker provides real-time data acquisition.

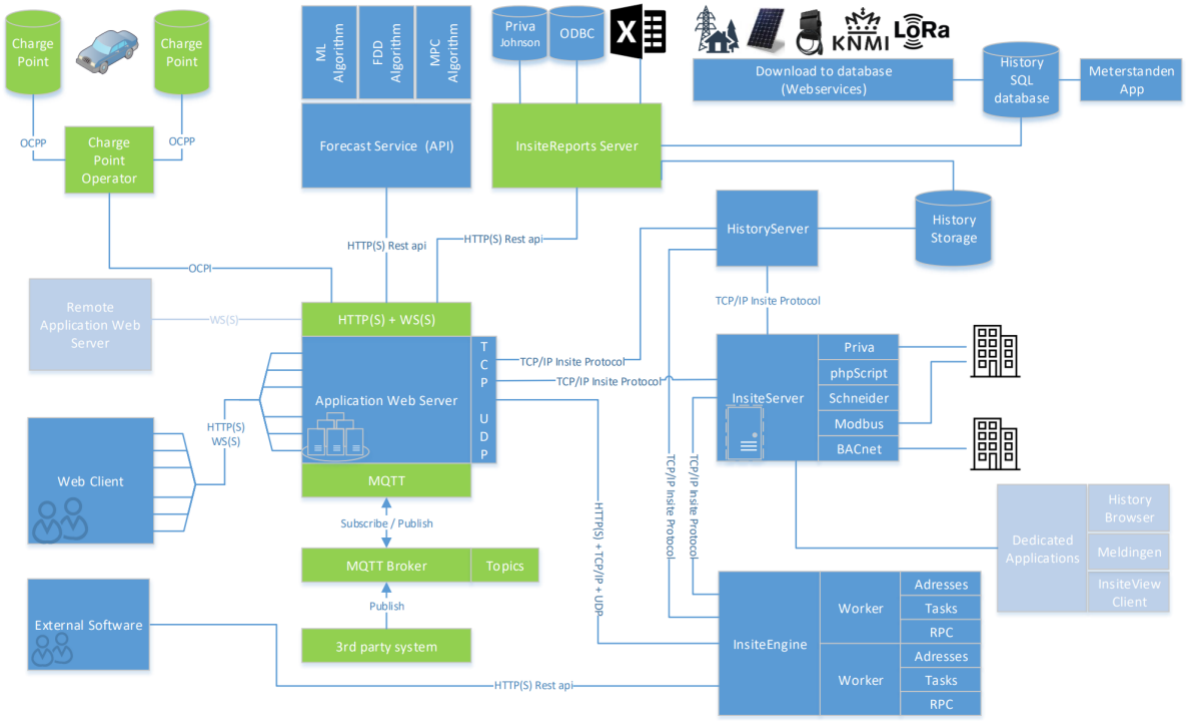

Figure 8 Kropman's InsiteView system architecture (image used with permission from Kropman)

2.6 Span of Reference Architectures

Some reference architectures are more abstract and high-level, while others are closer to technical implementation. Abstract reference architectures typically provide a conceptual overview of the system, its components, and how they interact with each other. They focus on the system's functional requirements, non-functional requirements, and the flow of data and control within the system. These types of reference architectures are often used as a starting point for system design and provide a common language for stakeholders to communicate and align on the system's vision and goals.

On the other hand, more technical reference architectures provide a detailed description of the system's technical components, their interfaces, and how they are implemented. These architectures often include information on the hardware and software used, the protocols used for communication between components, and the specific services provided by each component. A technical reference architecture is often used by developers and engineers to build and deploy the system.

Figure 9 shows the reference architectures we discussed above, in a scale of more abstract to low level detailed architectures.

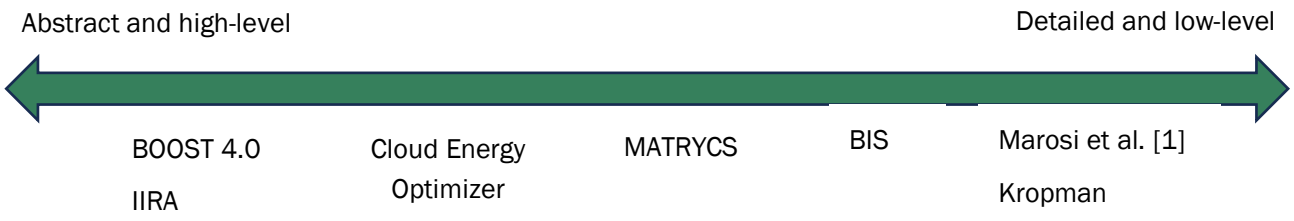

Figure 9 Reference architectures with various abstraction levels

[8] https://ivs.kropman.nl/insiteview/

2.7 Summary

Some notable software reference architectures including the MATRYCS-A Big Data Architecture, the BOOST 4.0 Reference Architecture, and the IIRA (Industrial Internet Reference Architecture) were discussed in this section. Reference architectures are pre-designed solutions that provide a blueprint for building complex systems. They define a set of standards, guidelines, and best practices for designing, building, and operating systems in a specific domain. They can help reduce development time and costs, improve system quality, and facilitate interoperability. It is important to note that reference architectures are not one-size-fits-all solutions and may require customization to meet specific system requirements and business needs, that in turn imposes time constraints and hence guides technical choices. Four key features identified from these reference architectures are:

1. Developed with a specific domain in mind (such as IoT, manufacturing, data analytics)
2. Have logical layers to group similar functionalities
3. Can be high level (more abstract) or more elaborative (low level)
4. Implementation is optional

3 DESIGN CONSIDERATIONS

This chapter aims to present the design considerations of the proposed Reference Architecture. We consider *stakeholders* and their *concerns* as an input to derive a set of functional requirements to design the Reference Architecture. Non-functional requirements of the Reference Architecture are derived using best practices in software system design.

3.1 Stakeholders and their concerns

There are a few stakeholders who have concerns for smart building systems such as,

- Facility managers,
- Building owners,
- Third party service providers (Building engineers, Software platform developers (e.g. BMS providers), Original Equipment Manufacturers (OEM)), and,
- Building occupants

A detailed discussion on the various data needs and requirements in smart buildings concerning various stakeholders (mainly building owners and third-party service providers) is available in our previous deliverable *D4.03 Data Needs and Requirements in Smart Buildings*[9]. However, some of the concerns are described here. Facility managers, and third-party service providers (Building engineers, Software platform developers) are the stakeholders that mostly interact with data. One of the primary concerns is extracting data from heterogeneous systems, which involves collecting data from different sources and making it available for various analytics. Periodically downloading external data such as weather from external APIs and the ability to upload various timeseries data sets and contextual data sets in 2D and 3D formats are also important requirements. These data sets then need to be combined and processed for analysis, which involves tasks such as data cleaning, pre-processing, and integration. Ultimately, actionable strategies for goals such as energy efficiency must be created, and analytics results must be executed in controllers and linked to digital twin platforms or visualization environments for effective monitoring and control. Building owners, on the other hand, are interested in comparing performance of different buildings in a portfolio and identify energy efficiency opportunities. For this, they wish to interact with different user interfaces that provide them with the necessary tools.

Various smart building applications were studied in our previous deliverable 'D4.03 on Data needs and requirements in smart buildings'. We identified the five most often encountered smart building applications as:

1. Fault detection and diagnostics (FDD)
2. Occupancy and occupant behaviour prediction
3. Energy flexibility and Demand Side Management (DSM)
4. Efficient HVAC control
5. Facility management and monitoring building performance

In general, all the applications under these categories need data collected from the physical systems via sensors, perform various data analytics to identify optimum control strategies and perform these strategies via actuators in the building (Figure 10). These applications consume a large amount of heterogeneous data and a system architecture should enable efficient means to do so.

[9] https://brains4buildings.org/wp-content/uploads/2022/05/B4B-WP4-D4.3_Data-Needs-and-Requirements.pdf

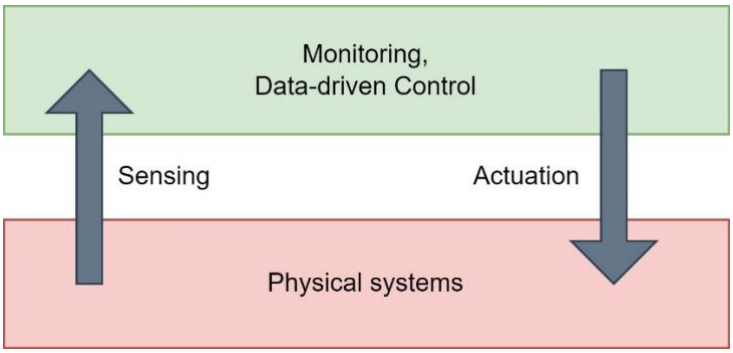

Figure 10 Data flow in smart building applications

Based on the concerns of stakeholders and the nature of smart building applications, a set of functional requirements are derived. These functions can be categorized into six categories, which are identified as functional views. These functional views are:

1. Data collection
2. Data management
3. Data analytics
4. Communication
5. Control
6. Visualization and monitoring

Figure 11 shows the relationship between the stakeholders, functional requirements and functional views.

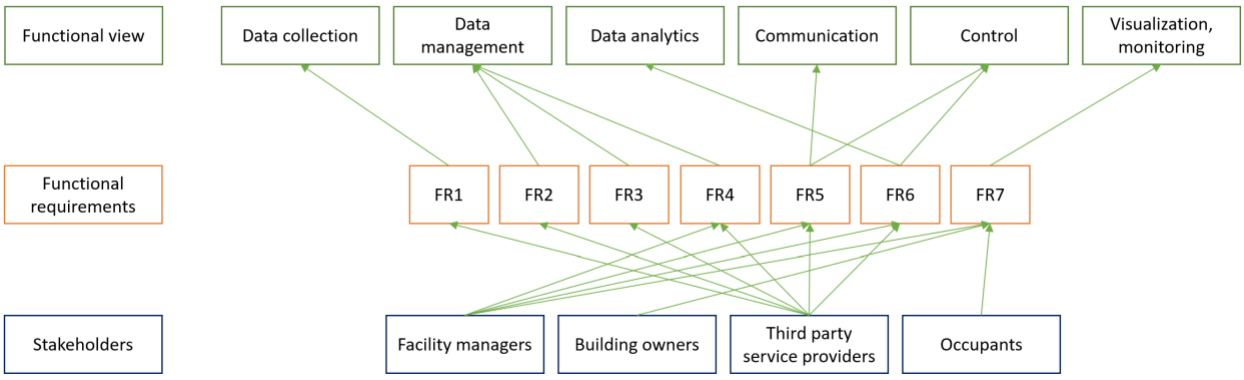

Figure 11 Relationship between stakeholders, functional requirements and functional views

3.2 Functional requirements

Functional requirements define what the system architecture should do. We derived seven functional requirements by referring to the concerns from stakeholders as described above, smart building literature [5], [12], [13], and our own experience in the Brains for Buildings project (living labs).

1. FR1 - Ingestion of time series data from heterogeneous systems.
2. FR2 - Gathering, management, and processing of large volumes of time series data.
3. FR3 - Gathering, management, and processing of contextual information (metadata) from various data sources.
4. FR4 - Integrating metadata with the operational data of various devices in a building.
5. FR5 - Interoperability among the several existing devices, services, and applications.
6. FR6 - Real-time or near real-time bi-directional communication ability and the ability to autonomously control facilities within the building.
7. FR7 - User-centered interfaces for easy data access.

3.3 Non-functional requirements

Non-functional requirements are characteristics or qualities that a system or software must possess, but are not directly related to the system's functionality. Non-functional requirements define the system's overall behavior, performance, and quality attributes and are essential to ensure that a system is reliable, secure, and efficient in performing its intended functions identified under functional requirements. We derived seven non-functional requirements based on the literature [12] and our own experience in Brains for Buildings project (living labs).

1. NFR1 - Modularity
2. NFR2 - System availability (load time)
3. NFR3 - Efficiency (speed and performance)
4. NFR4 - Interoperability
5. NFR5 - Security
6. NFR6 - Usability
7. NFR7 - Scalability and extensibility

Modularity: The ability of the smart building system architecture to be broken down into separate, independent modules that can be developed, tested, and maintained separately. This helps to ensure that changes or updates to one module do not affect the entire system. Modularity also has an impact on scalability and extensibility of the system.

System availability: Refers to the ability of the smart building system architecture to remain operational and accessible to users over time. This includes factors such as uptime, system response times, and fault tolerance.

Efficiency: Designed to use resources (such as power and processing) in an optimal way. This can include considerations such as load balancing, data compression, and energy-efficient hardware.

Interoperability: Designed to work seamlessly with other systems and technologies, both within and outside the building. This can include protocols and standards that enable interoperability, as well as APIs for integrating with other systems.

Security: Designed with strong security measures to protect against unauthorized access, data breaches, and other cyber threats. This can include encryption, authentication, and access control mechanisms.

Usability: Designed to be easy to use and understand for building occupants, maintenance staff, and other stakeholders. This can include intuitive user interfaces, clear documentation, and training resources.

Scalability and extensibility: Designed to support growth and expansion over time, including the addition of new features, systems, or technologies. This can include strategies such as modular design, flexible data structures, and support for open standards

4 REFERENCE ARCHITECTURE

This chapter of the report outlines the proposed system architecture, which comprises five layers: Data Sources, Data Integration, Data Management, Application Logic, and Visualization. Figure 12 shows a block diagram of the proposed system architecture. Functional requirements (FR) under each functional domain are realized within suitable layers of this architecture using microservices.

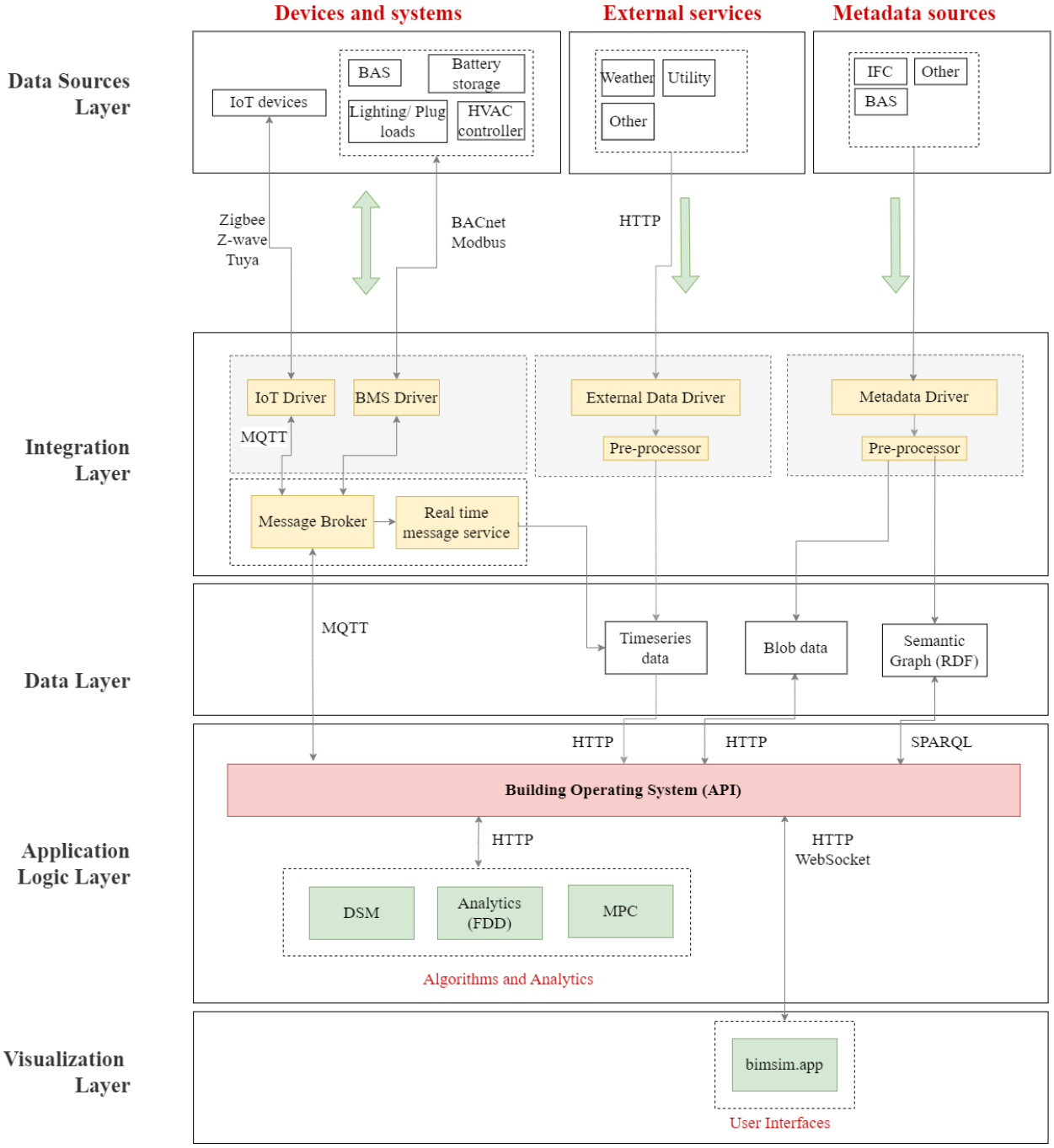

Figure 12 Five-layer microservice-based reference architecture

The primary objective of these layers is to improve the system's logical separation of concerns. Each layer consists of multiple containerized microservices that provide specific functionalities.

The Data Sources layer is responsible for collecting data from various sources, including BMS, IoT devices, external APIs, databases, and data streams. The Data Integration layer combines this data and transforms it into a format that the Data Management layer can use. The Data Management layer stores and manages the data to ensure consistency, availability, and security. The Application Logic layer implements the system's business logic (smart building applications like FDD and MPC), utilizing the data from the Data Management

layer. The Visualization layer presents the system's results in a user-friendly and understandable manner, providing a graphical interface that enables users to interact with the system's functionalities and visualize the data. In the upcoming sections, each layer is explained in detail.

4.1 Data sources Layer

The proposed system architecture includes a Data Sources Layer that comprises three types of data source categories: a) devices and systems, b) external data sources, and c) metadata sources.

A) Devices and systems

The devices and systems source real-time data on various systems, including indoor air quality parameters, valve positions, and temperature values, which are continuously measured using different sensors connected to a BAS system. This time series data is collected via BAS systems and stored in databases for analytical purposes. In addition, buildings also use IoT devices for various use cases, such as monitoring indoor air quality parameters and smart lighting, generating a large amount of operational data in the form of time series data. This data also needs to be collected via messaging services such as Pub/Sub that relies on protocols like MQTT.

B) External data sources

External services such as utility companies and weather services provide data that energy flexibility and demand-side management applications need to access. This data is periodically downloaded and analyzed in combination with sensor data or connected to via relevant APIs.

C) Metadata sources

Metadata sources include BIM models, P&ID diagrams, metadata dictionaries, and manuals that contain valuable information about the building, its systems or data schemas used in various data platforms. However, the metadata in these sources is expressed using different classification methods (such as key-value, IFC), making it challenging to integrate them with each other or with operational data from devices and systems. Sensor metadata in buildings is typically available as a metadata tables, which provides additional information about the sensor, such as its location and type. These tables are produced by extracting relevant fields from the BAS systems, or scanning the building automation network. BIM models are another essential metadata source, as they contain contextual data that is structured and machine interpretable. BIM models can be created using proprietary or open BIM authoring tools and converting them to a vendor-neutral file format, such as the IFC data standard, may be necessary. The IFC data standard is an open standard that improves information exchangeability between BIM software applications and provides a classification system that standardizes metadata for smart building applications.

4.2 Integration Layer

Achieving integration and interoperability in smart buildings can be achieved through several approaches.

1. Standards-based Approaches: Adopting industry standards, such as BACnet, Zigbee, and Z-Wave, can help to ensure that different components and systems within the smart building can communicate and exchange data effectively. However, communicating with other services that do not follow the same standard cannot be achieved by only relying on this approach, and need middleware/drivers that translates one standard into another.
2. Middleware/Drivers: Using middleware [2], such as gateways and brokers, can help to mediate communication and data exchange between different components and systems, ensuring that the data is in the appropriate format and that the systems can work together seamlessly.
3. APIs and Protocols: Implementing open APIs and protocols, such as REST and MQTT, can help to facilitate integration and interoperability between different systems and components within the smart building, as well as with external systems and technologies.

Therefore, the Data integration layer consists of microservices that are required to integrate data incoming from the data sources layer and facilitate the bi-directional communication using middleware/drivers. Depending on the nature of the data source, there are several services in the Integration layer that can be utilized. The integration layer provides drivers or adapters to connect to the data streams of the data sources.

A suitable driver is required for each data source depending on the underlying protocol, data format, and how the data is exposed. The ability to integrate different data sources on demand is key to extending the possible applications that can be implemented atop the system architecture.

Some essential drivers are described below. These drivers or data integration methods are subdivided in three categories, and each category is linked to its Data type in the Data sources layer.

A) Integrating operational data of devices and systems

BMS Drivers

The BMS driver is the interface between the BMS and the system architecture, providing access to data from the BMS by connecting to an existing time series database or by loading available historical data to a time series database. Communication with the BMS controllers is usually based on protocols like BACnet, but communication within the proposed architecture is implemented using REST APIs and a message broker. Therefore, a translation mechanism to the BACnet protocol should be implemented in these drivers. Example implementations of such drivers can be seen in Figure 6 and Figure 8.

IoT Drivers

For IoT devices, we propose a driver where sensor data acquired from different IoT sources is published into a central message broker. Whenever a new device needs to be integrated, a new IoT driver must be introduced. Real-time data available via this MQTT broker can be accessed through a message broker, such as Mosquitto, using the MQTT protocol.

Message Broker

The message broker is used for two purposes. The first one is to bring the real-time message streams coming from multiple IoT services to a central location. The second service provided by the message broker is listening to the control commands published by various applications. For example, an application in the application logic layer can publish a message with a control command to change the set point of an HVAC controller. When the broker receives this message, the BMS driver which listens to the control command topic of that device can execute this command.

B) Integrating contextual data of metadata sources

To address the lack of standardization for representing metadata of operational data in buildings, smart building ontologies [15] and tagging systems are being introduced. Integrating contextual data of metadata sources is done using smart building ontologies that create a semantic model of the building, also known as a metadata schema [16]. Linked data techniques are used to connect data from different domains. To combine multiple ontologies in the Architecture, Engineering, and Construction (AEC) and smart building domains, ontology alignment and mapping techniques are used to identify common concepts like a space/room [17]. Operational time series data is not integrated into these metadata schemes, but the reference to time series data is included [18], enabling the automatic querying and retrieval of time series data associated with the relevant statements in the metadata schema.

These ontologies provide a formal representation of the domain's concepts and relationships, classes, and attributes, using a graph data structure in which concepts are represented by graph nodes, and relationships are represented by graph edges. An example from the Brick ontology[10] is shown in Figure 13.

[10] https://docs.brickschema.org/intro.html

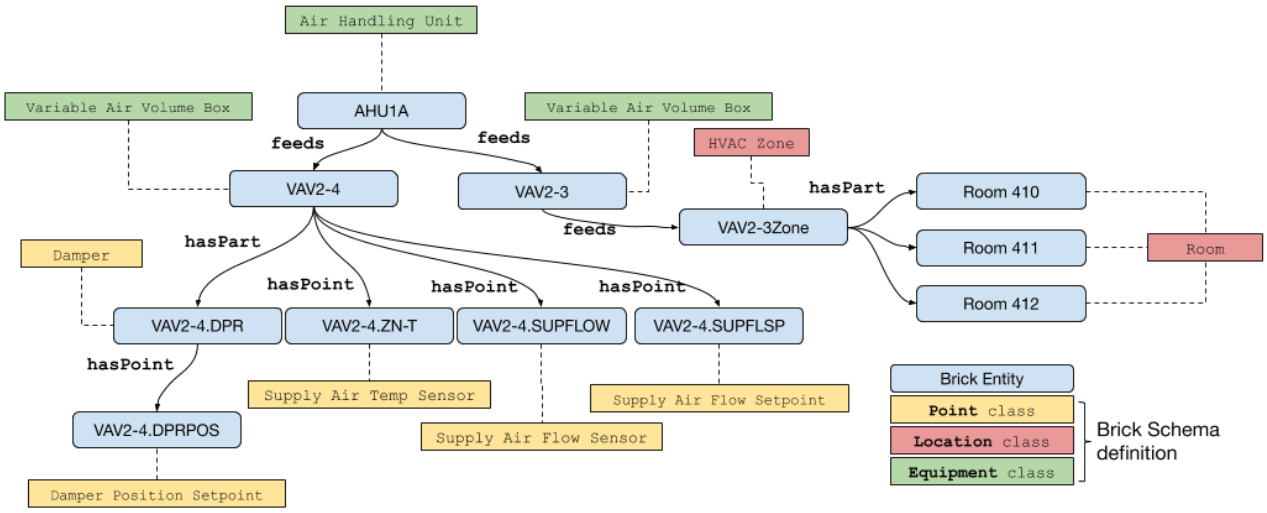

Figure 13 Representation of HVAC system using the Brick ontology

To generate a metadata schema, available metadata is converted into subject-predicate-object triples that conform to the used ontologies. This conversion again requires multiple drivers, depending on the source system, with essential domains being the building topology, sensors, and observations. Two types of metadata drivers are implemented, one for the IFC-to-RDF conversion that handles the building topology, and another for generating a metadata schema based on BMS and IoT sensors using Brick and other Linked Building Data (LBD)[11] ontologies.

IFC-to-RDF converter

Creating a metadata schema from an IFC model requires a purpose-built converter to understand the available resources and map them to a desired ontology. An IFC-to-RDF converter[12] is used to generate the semantic model of the building based on the IFC file, transforming the IFC statements into RDF statements conforming to the PROPS, and BOT[13] ontologies.

BMS metadata connector

Semantics are extracted from BMS metadata sources like data dictionaries and device specifications, and a mapping table is used to identify sensors based on their naming convention. The Brick-builder tool[14] is used to parse the file and manually configured relationships to create an RDF graph. The Brick and SSN[15] ontologies are used to describe the BMS and IoT domains. IoT data streams are identified by their *topics*, which act as the unique identifier of the real-time sensor data of an IoT node and can be included using a *brick:hasTimeseriesId* relationship. The metadata schema is stored in a graph database like GraphDB[16] to enable querying and integration with other services.

4.3 Data Management Layer

To enable efficient storage and querying of various types of data, the proposed system architecture must support different data formats, including time series data, RDF data, object data, and user data. Each data type requires specific database technology for optimal performance. For example, time series databases are best suited for storing and querying time-based data, while object databases are ideal for storing files. Object-oriented storage can also be used for archiving large files and sensor data archives to support machine learning applications that require significant amounts of data. User and project details for various applications

[11] https://github.com/w3c-lbd-cg/ontologies

[12] https://github.com/pipauwel/IFCtoLBD

[13] https://w3c-lbd-cg.github.io/bot/

[14] https://docs.brickschema.org/lifecycle/creation.html#from-structured-tabular-sources

[15] https://www.w3.org/TR/vocab-ssn/

[16] https://www.ontotext.com/products/graphdb/

can be stored in either relational or document databases. RDF graphs can be stored in GraphDB, a specialized graph database designed for optimal graph data storage.

4.4 Application Logic Layer

The Application Logic Layer comprises of the REST API and smart building applications.

A. REST API

The REST API acts as an intermediary between microservices within each layer, including web applications, databases, and real-time messaging servers. Through the API, historical sensor data can be queried using relevant unique sensor IDs. Additionally, the API provides endpoints for querying the RDF database and project data stored in a document database. Main endpoints of the API are shown in Figure 14.

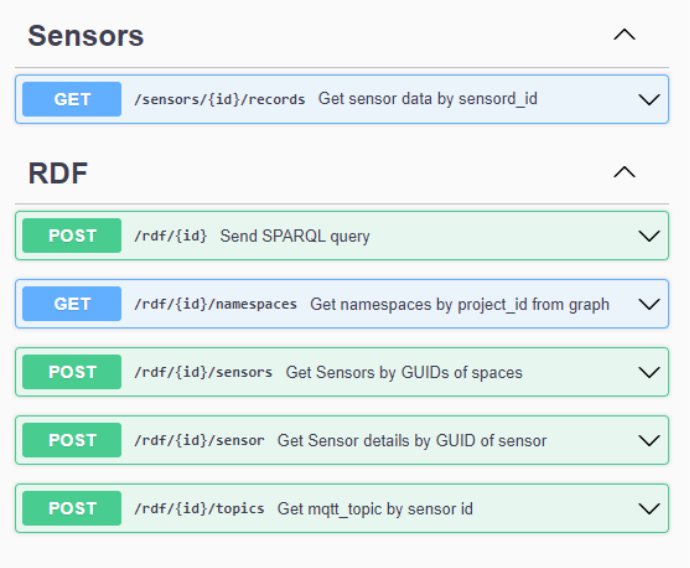

Figure 14 Main endpoints of the API for accessing timeseries database and graph database

B. Smart building applications

Smart building applications are microservices that combine data-driven algorithms like FDD, energy flexibility and the BAS system. The proposed microservice architecture facilitates the integration of these applications with the existing data sources via API communication. For a closed-loop system, data collection, algorithm development, and control strategy implementation are accomplished by exchanging data through the API and message broker.

An example implementation of an MPC for an energy flexibility application with the proposed reference architecture can be mapped to following steps.

1. *Data Collection:* The first step is to collect data from the various sensors and systems within the smart building, such as temperature sensors, occupancy sensors, and energy usage meters. This data is used as input to the MPC. These data are obtained via various drivers (BMS, weather) in the Data integration layer from the Data sources layer and recorded in the timeseries database in the Data layer.
2. *Data Pre-processing:* The data collected from the sensors and systems needs to be pre-processed and transformed into a format that can be used by the MPC. This may include cleaning and normalizing the data, as well as transforming it into a form that can be used for predictive modeling. This step happens within the MPC microservice developed within the Application logic layer within a development environment such as Python/JupyterNotebook. Here, required data can be obtained from databases by sending API requests with necessary sensor IDs.

3. *Predictive Modeling:* The next step is to build a predictive model that can be used to forecast the future state of the HVAC system based on the input data. This may involve using machine learning algorithms, such as neural networks or decision trees, to develop a predictive model. This step also happens within the MPC microservice developed within the Application logic layer within a development environment such as Python/JupyterNotebook.
4. *MPC Implementation:* Once the predictive model is built, the MPC can be implemented to control the HVAC system. The MPC takes the input data and uses it to make predictions about the future state of the system. The MPC then uses these predictions to optimize the control of the HVAC system in real-time, ensuring that the system operates efficiently and effectively. Outputs of the controller can be published to the real-time message broker (like MQTT). Thereafter, the suitable drivers in the Integration layer translates these setpoints/commands to the device protocols in the Data sources layer.

Example applications developed according to the proposed architecture are demonstrated in Chapter 5.

4.5 Visualisation Layer

The Visualization Layer comprises of interfaces, web pages, dashboards, or charts that can provide meaningful ways to communicate information to the end-user. The choice of which interfaces to use depends on the type of application.

4.6 Implementing non-functional requirements

In the previous chapter, seven non-functional requirements were introduced as:

1. NFR1 - Modularity
2. NFR2 - System availability
3. NFR3 - Efficiency
4. NFR4 - Interoperability
5. NFR5 - Security
6. NFR6 - Usability
7. NFR7 - Scalability and extensibility

How these non-functional requirements can be integrated with the reference architecture is discussed below.

Modularity: Modularity of the system can be achieved by a service-oriented approach. This means that each new application or service is implemented aa a microservice and the communication between this and other services is realized using the API and message broker. This way, new components can be added or removed without affecting other components. Software tools such as Docker[17] and Kubernetes[18] can be used to containerize and orchestrate modular components of a smart building system.

System availability: Availability refers to the ability of the smart building system architecture to remain operational and accessible to users over time. This includes factors such as uptime, system response times, and fault tolerance. Tools such as Prometheus[19] can be used for monitoring the availability of system components and alerting them when problems occur.

Efficiency: Efficiency can include considerations such as load balancing, data compression, and energy-efficient hardware. Software tools like Apache Spark, Hadoop, and Apache Flink can be used for big data processing and analytics to improve system efficiency.

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

[18] https://kubernetes.io/

[19] https://prometheus.io/docs/introduction/overview/

Interoperability: Communication between different components should be achieved by API and message brokers. Middleware software like Apache Kafka[20], Mosquitto or RabbitMQ[21] can be used to enable interoperability between different system components and protocols.

Security: For authentication and access control, libraries such as openID[22] which implement OAuth2.0[23] specification can be used. For secure connection between the building and the remote cloud, VPN connections can be established using VPN solutions like OpenVPN[24].

Usability: Depending on the use case, different user interfaces for interacting with the building data are needed. Some examples include Grafana dashboards, web applications and digital twin platforms.

Scalability and extensibility: To support growth and expansion of resources over time, cloud-based infrastructure can be used. Cloud-based platforms like Amazon Web Services (AWS) and Microsoft Azure can provide scalable and extensible infrastructure for smart building systems, allowing for easy expansion and growth. Typically, there are three main cloud service models to choose from:

1. Infrastructure as a Service (IaaS): This model provides the most control and flexibility over the cloud infrastructure. The smart building can choose to use virtual machines, storage, and networking resources as per its requirements.
2. Platform as a Service (PaaS): This model provides a platform to build and deploy applications without worrying about the underlying infrastructure. This can be a good option for smart buildings that need to quickly develop and deploy new applications.
3. Software as a Service (SaaS): This model provides complete software solutions that can be accessed over the internet such as databases and message brokers. This can be a good option for smart buildings that want to use off-the-shelf applications without having to worry about infrastructure and maintenance.

[20] https://kafka.apache.org/

[21] https://www.rabbitmq.com/

[22] https://openid.net/what-is-openid/

[23] https://auth0.com/intro-to-iam/what-is-oauth-2

[24] https://openvpn.net/

5 IMPLEMENTATION

A reference architecture is implemented in one of the Brains4Buildings Living Labs, and smart building applications are continuously being tested using this implementation. In this section, we discuss the current implementation details and the smart building applications we tested during the implementation of this architecture. The main Living Lab deployed for testing this reference architecture is the Atlas Living Lab at Eindhoven University of Technology.

5.1 Software implementation

This section describes the software components used to implement an instance of the reference architecture.

Data sources layer:

- In the current implementation we have replicated a database to represent the BMS. We used three IoT sensors to test real-time data streams.
- At this stage, we do not use external data sources like weather or utility signals, and this will be implemented in future.
- We use BIM models and BMS metadata tables under the Metadata sources category.

Integration layer:

- We implemented three IoT drivers to subscribe to their real-time data streams. These drivers are Node.js services. We used Eclipse Mosquitto as the message broker. Real-time message service is another Node.js service that unifies IoT payloads into a flat map structure.
- For metadata integration, we implemented the two metadata connectors described in the previous chapter, IFC-to-RDF converter and the BMS metadata connector. There services run as standalone applications and are not integrated into the system yet.

Data layer:

- We instantiated three databases to facilitate different types of data. Timeseries data is stored in a MongoDB Timeseries collection. Files such as BIM models are stored in MINIO, a blob storage database solution. Semantic graphs are stored in a GraphDB graph database in RDF.

Application logic layer:

- This layer consists of the API and the smart building applications. We implemented a Node.js API using Nest.js framework and created several endpoints to interact with the three databases.
- At this stage, there are only monitoring and no control applications are implemented. These applications are described in the next section.

Visualization layer:

- Visualization layer consists of a web application developed using React.js framework. This web application was developed to visualize a BIM model in the browser and link the timeseries data of sensors in to the Rooms in the BIM model.
- We also implemented a Grafana dashboard to query sensor data in combination with a Brick model of a building to demonstrate the metadata integration process.

  These two applications are described in the next section.

5.2 Implemented smart building applications

In this section, we demonstrate the applications we have implemented so far using the proposed architecture.

A) BIM IoT Integration

This application is designed to integrate timeseries data of sensors with a BIM model of a building. The web application provides access to the BIM model via a browser and is used to demonstrate the functionality of the proposed system architecture. The REST API endpoints are utilized to connect the web application to the semantic graph, time series data and BIM models. Full description of the implementation is published in the article "A web-based approach to BMS, BIM and IoT integration: a case study. CLIMA 2022 Conference"[25].

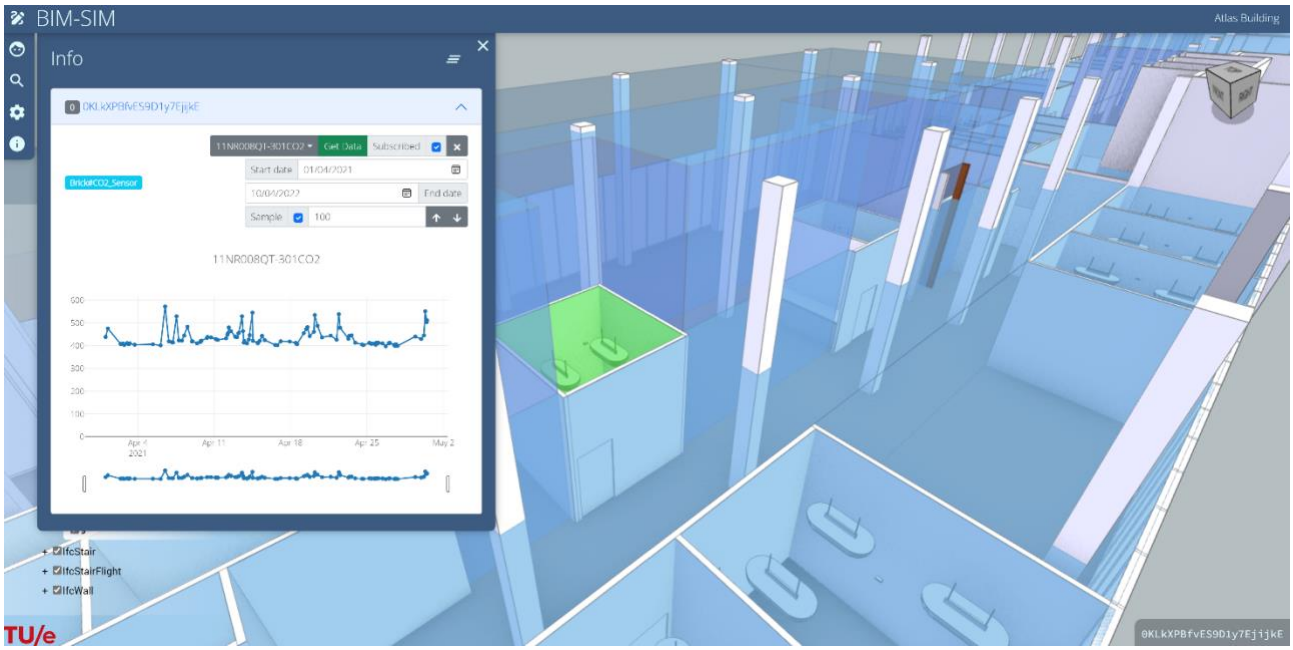

Figure 15 Visualizing BIM model and timeseries data using the web application chamari [18]

B) Real-time message service

This application was developed for standardising the payloads of different formats coming from multiple IoT devices into a common format and allow front-end applications to connect to these real-time data streams. The Atlas Living Lab of TU Eindhoven is used as the example building. A metadata schema of the building was created using the metadata drivers of the proposed architecture. To test the real-time data streams, three IoT sensors were used. The goal is to filter the available IoT sensors in the building according to their location, and visualize their real-time data streams of the building using a web application. The intended functionality is to query the building's structure first (as in floors and rooms), and then select a sensor that belongs to the particular room to see its real-time data stream.

Figure 16 illustrates the process of data acquisition from different IoT devices via Mosquitto MQTT Broker, processing them (flat map) to a unified structure and publishing them to a message broker. This application comprises five components; 1) a collection of data acquisition modules (data providers), 2) a module for unifying schemata and semantics (message processor), 3) Pub/Sub and real-time streaming service (based on MQTT and WebSocket), 4) an ontology-based information model and 5) the API.

The first component is a collection of data acquisition modules, which are responsible for collecting data from various sources. The second component is a message processor that unifies schemata and semantics to

[25] https://proceedings.open.tudelft.nl/clima2022/article/view/228

ensure consistency and interoperability. The third component is a Pub/Sub and real-time streaming service that is based on MQTT and WebSocket protocols, which enables seamless data exchange between the browser and server. The fourth component is an ontology-based information model that defines the data structure and relationships between different data elements. Finally, the fifth component is the API that provides a programmatic interface for external applications to access the data. This architecture is designed to provide real-time sensor data to web applications, and hence, the use of a WebSocket protocol is emphasized. The WebSocket protocol enables continuous bi-directional data exchange between the browser and server, which is essential for applications that require constant data exchange without continuously sending requests to a web server.

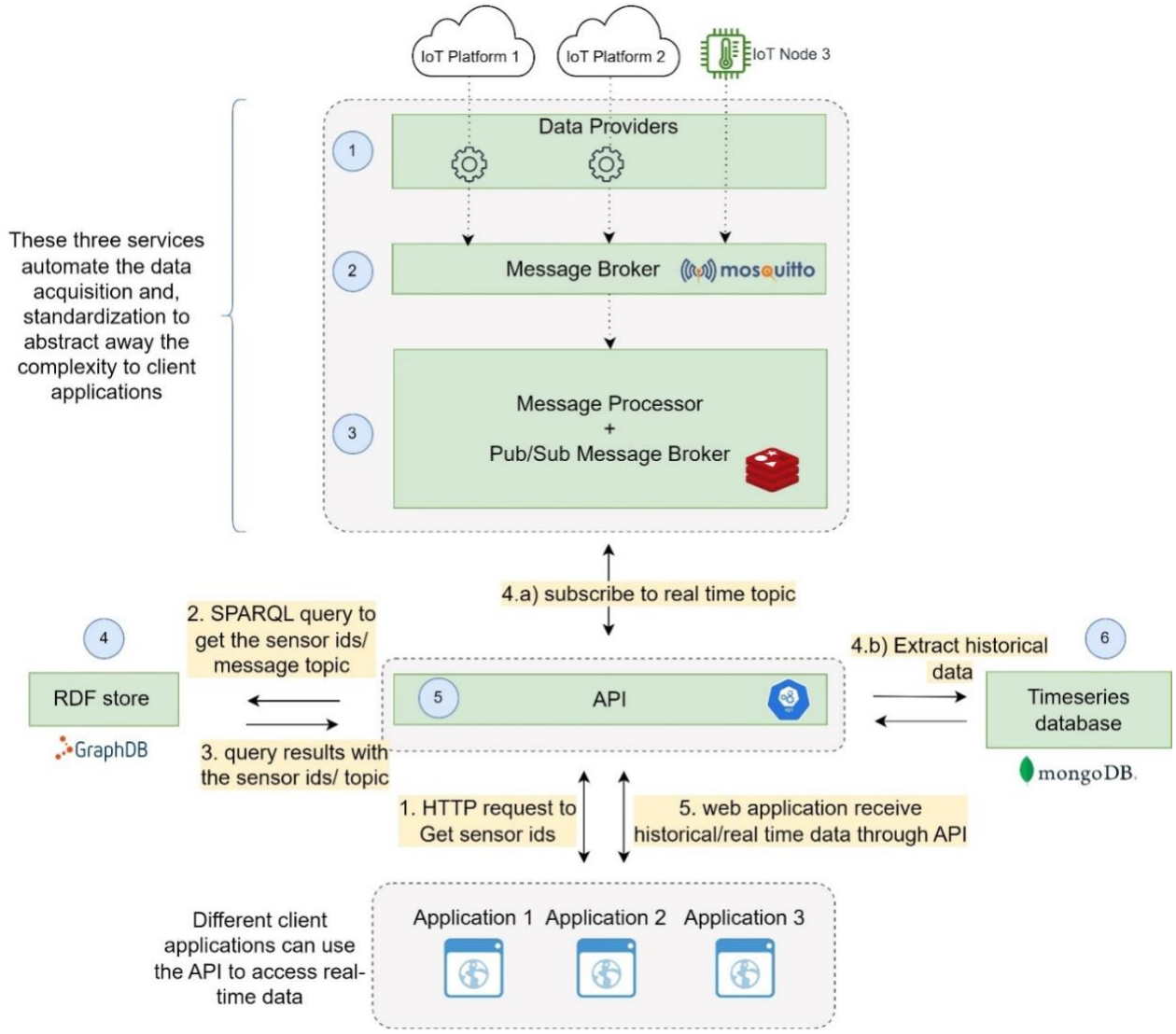

Figure 16 Connecting to real-time message streams of IoT devices

Data providers

A collection of data providers is utilized, with each provider functioning as an independent microservice. This microservice-based approach allows for the implementation of data connectors for various IoT devices, which can operate independently based on their unique protocols and data models. Furthermore, the addition or removal of a data provider will not affect the other providers. The primary function of the data provider is to acquire IoT sensor data from different platforms and publish it to the Pub/Sub message broker. Each IoT sensor node is identified by a unique topic, and the data from that node is published to that topic. The current implementation employed three types of IoT sensor nodes:

1. Indoor Air Quality sensor from Edimax, measuring CO_2, temperature, humidity, CO, TVOC, PM2.5, PM10, and HCHO. Sensor data can be accessed hourly, daily, weekly, or in real-time via their API.

2. Smart socket from BlitzWolf, measuring voltage, current, and power. Sensor data can be accessed via their API.

3. Custom built ESP32-based sensor node, measuring temperature, humidity, CO_2, TVOC, illumination, and battery voltage.

Pub/Sub message broker

The proposed framework employs a Pub/Sub messaging pattern, where a message published by a sender to a specific topic is instantly received by all subscribers subscribed to that topic. The sender publishes the message for a topic rather than a particular receiver, and the broker manages the connection between the publisher and subscribers. In this study, the message broker used is Eclipse Mosquitto, which is based on the MQTT protocol.

Message processor

The message broker used in the proposed framework receives messages in the original format published by each IoT node, resulting in messages in various formats, such as nested and flat maps. To ensure uniformity, the message processor service converts these messages into non-nested *key:value* pairs. The processed messages are then republished to a message broker. For this purpose, the Redis Pub/Sub broker is utilized. As the proposed framework is designed to manage data for web-based applications, the WebSocket library, socket.io, is used. A sample of a processed message is shown in Figure 17.

```
{
  topic: 'esp32/083AF266DD84/pub',
  keys: [ 'co2', 'tvoc',
  'te', 'rh', 'lux', 'vbat' ],
  values: [ 872, 71, 27.3233,
  56.39648, 586.6666, 4.1569 ],
  timestamp: 1662298546131
}
```

Figure 17 IoT payload transformed in to a uniform format

Information model (metadata schema)

The proposed framework utilizes semantic descriptions of the information model using SSN/SOSA, Brick, and LBD ontologies. This enables accessing, integrating, and extracting information from various data providers (IoT and non-IoT such as BIM, BAS, etc.). A sample of the information model is shown in Figure 18, while the complete model is available in the GitHub repository[26]. The Brick ontology's time series identification is used to incorporate MQTT topics, allowing unique identification of real-time data from an IoT node located in a specific space in a building.

[26] https://github.com/ISBE-TUe/atlas-building-graph

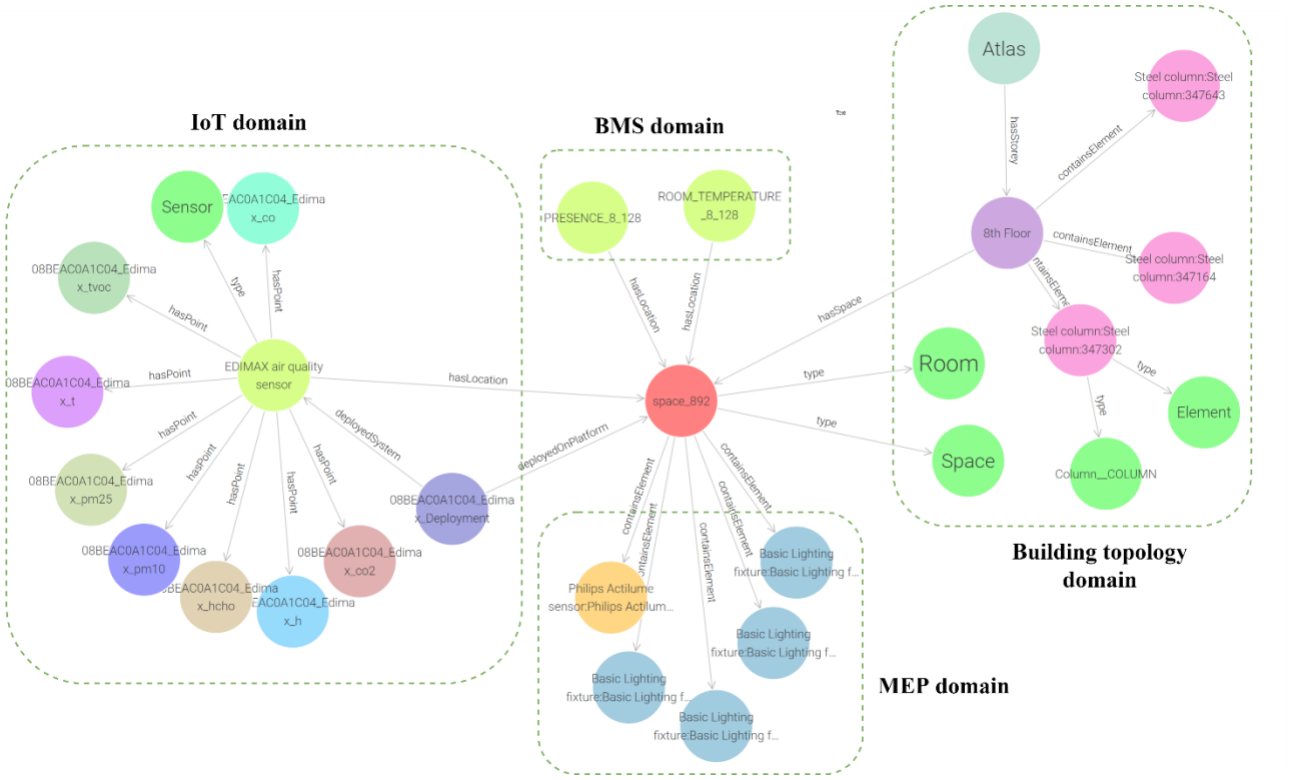

Figure 18 Part of the metadata schema of the building representing sensors and their relationships[27]

Application Programming Interface.

To fulfill client requests for data, a WebSocket connection is established between the API server and the client (web application). Additionally, the API endpoint for executing SPARQL Protocol and RDF Query Language (SPARQL) queries on the building and sensor information model. Figure 19 shows how the results of the queries on the metadata schema of the building are displayed in the web application. Figure 20 shows the real-time data stream of the selected IoT sensor.

[27] https://github.com/ISBE-TUe/atlas-building-graph

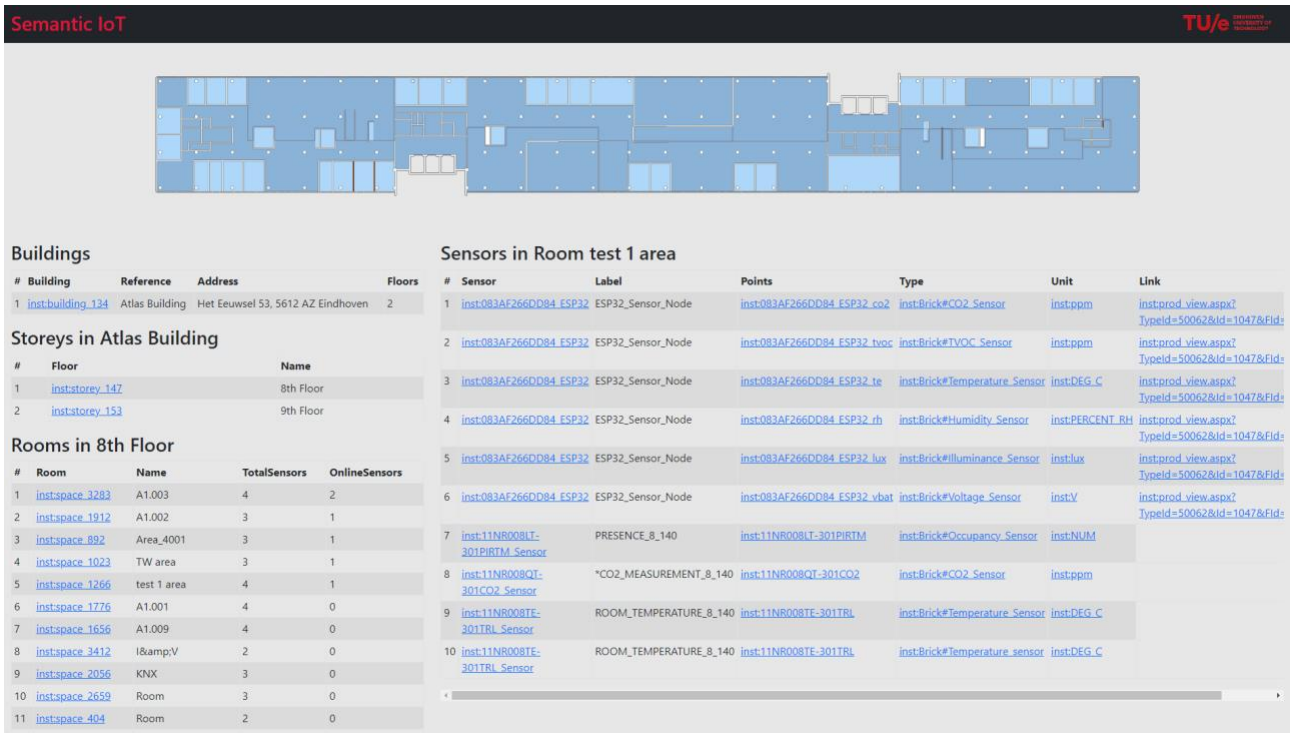

Figure 19 Visualizing the results of metadata schema querying for for the structure of the building and its available sensors

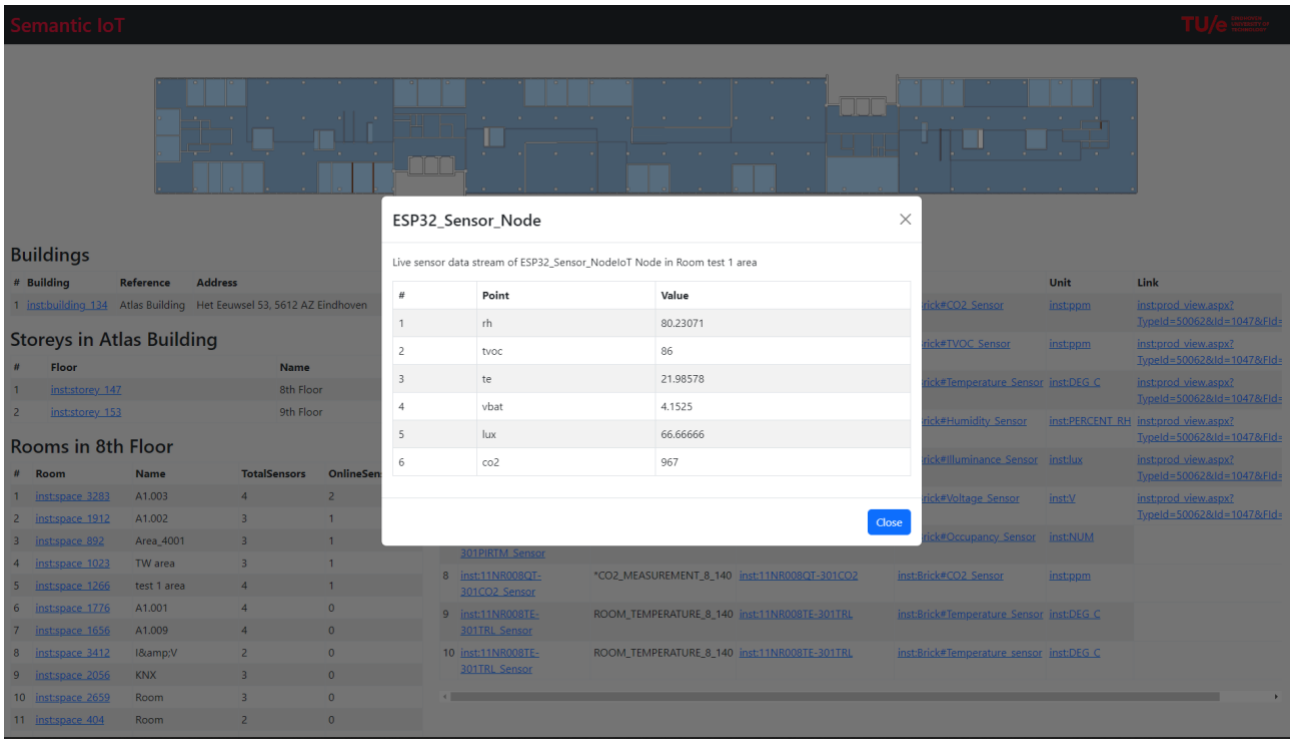

Figure 20 Connecting to and visualizing the real-time IoT data stream via the web application

C) Metadata schema generation

This application demonstrates the BMS metadata connector and the API. The building used for the demonstration is the Pulse building at TU Delft. We used the API to perform queries against the metadata schema and retrieve relevant time-series data from the time-series database. We also designed a Grafana dashboard to facilitate user exploration of metadata and time-series data. The API enables the communication between the time series database, GraphDB, and the Grafana web application.

Through the Grafana dashboard, users can filter the points of interest in the BAS by selecting equipment types from a dropdown list (green box in Figure 21). This list contains all the equipment types in the metadata schema generated. For instance, in Figure 21, *brick:Equipment* is selected as the equipment type *AHU*. This selection triggers a SPARQL query that requests the time-series identifiers related to all the points associated with *brick:AHU* in the graph. The resulting list of points is then displayed (red box in Figure 21), and users can select one or more of these identifiers to explore the time-series data. For example, in Figure 21, the *brick:Differential_Pressure_Sensor* has been chosen to explore the time-series data. This selection results in an API request being sent to the time-series database with the time-series identifier, and the charts are subsequently populated with data. Overall, this approach provides an efficient and user-friendly way of navigating through large amounts of time-series data with standard semantics.

Figure 21 Grafana application integrated with metadata schema of the building.

6 CONCLUSION

6.1 Mapping the design requirements to the reference architecture

In this report, we proposed a reference architecture for smart buildings. We started by analysing the various requirements of different stakeholders. Then we transformed these into functional and non-functional requirements for designing a system architecture. The design was informed by the literature, best practices and our own experience in the Brains for Buildings project. Here, the emphasis on the Application logic can be seen by the connection of three functional domains (data analytics, control, and communication) to this layer. The core of the system architecture lies within the Application logic layer.

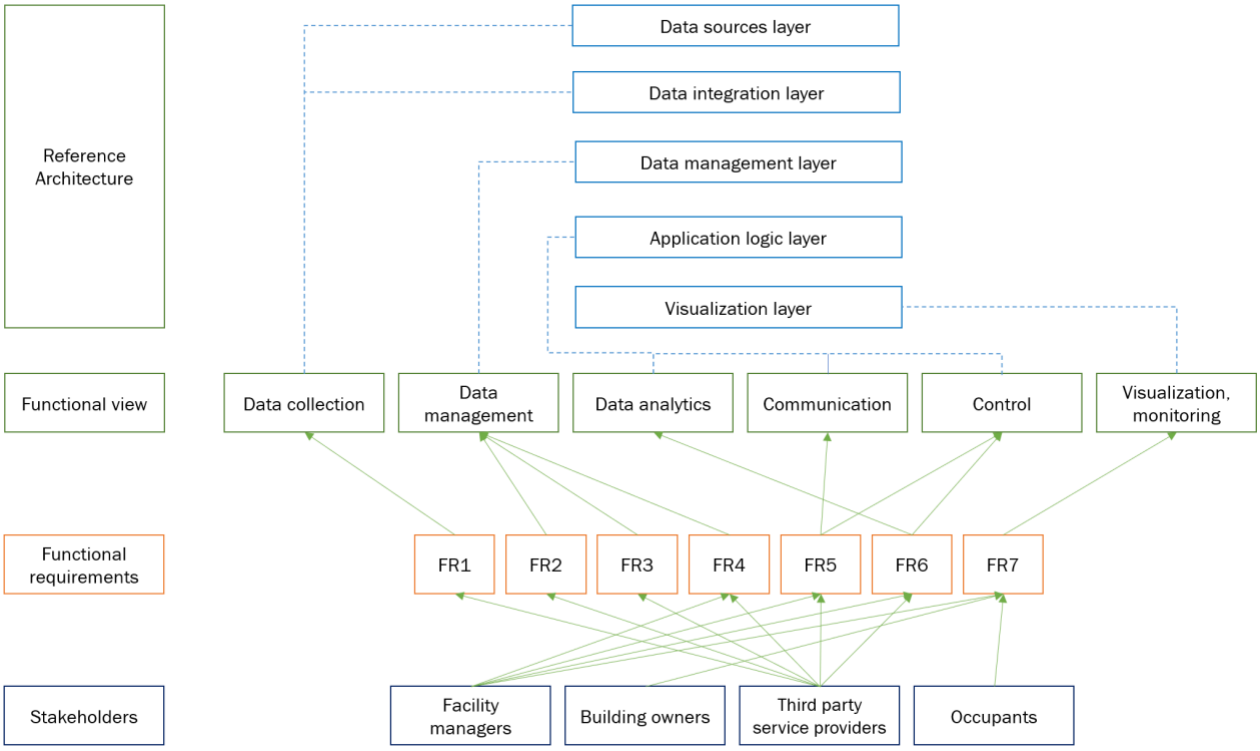

Figure 22 Mapping the relationship between stakeholders, functional requirements and functional views to the proposed reference architecture

6.2 Other technical considerations

There are other concerns regarding the optimum methods for data storage and usage which are interesting to investigate and further integrate with the proposed reference architecture. These are briefly addressed below.

6.2.1 Data retention period

The retention period for smart building data may vary depending on the specific use case and the type of data being collected. For example, sensor data for HVAC systems may need to be retained for a shorter period of time than occupancy data.

In general, it is recommended to keep the data for as long as it is useful for analysis and decision-making. However, longer retention periods may come with increased storage costs and potential privacy concerns. Therefore, it is important to consider the value of the data, the cost of storage, and any legal or regulatory requirements when determining the best retention period for smart building data. It may also be helpful to consult with domain experts and stakeholders to determine the appropriate retention period for specific use cases.

Some methods implemented by industry practitioners are:

1. Save only changes in values rather than all values
2. Save aggregated values rather than raw data
3. Discard the data after models are made

6.2.2 Optimum data storage

Several data storage techniques are suitable for smart buildings, depending on the data type and size, the required scalability and availability level, and other factors. Relational databases, Time-series databases, NoSQL databases, and Object storage databases are the most common. However, other concepts, including Data Lakes, Data Warehouses, Data Hubs, and multimodel databases, also exists, which may be suitable depending on the context. Therefore, the choice of database ultimately depends on the specific requirements and constraints of the application or system being developed.

6.2.3 Real-time vs non real-time control applications

A software system architecture for real-time smart building control applications should be designed with a focus on low latency and reliability in mind. Although many smart building applications are focused on batch analytics and generating insights, some applications like MPC need to interact with the building controllers in (near) real-time. For this, the system architecture must have:

1. Low latency: The system must respond with minimal delay to ensure efficient and effective building control.
2. Reliability: The system should be highly available and resilient, with failover mechanisms and backup solutions in case of system failures.

These are considerably different requirements compared to smart building applications that rely primarily on one-directional data analytics and information visualization. Specific care should be taken when implementing this solution, and this is expected to result in a system architecture that is much more custom than the reference system architecture presented in this document.

7 REFERENCES

[1] A. C. Marosi *et al.*, "Toward Reference Architectures: A Cloud-Agnostic Data Analytics Platform Empowering Autonomous Systems," *IEEE Access*, vol. 10, pp. 60658–60673, 2022, doi: 10.1109/ACCESS.2022.3180365.

[2] S. Dawson-Haggerty *et al.*, "BOSS: Building operating system services," *Proc. 10th USENIX Symp. Networked Syst. Des. Implementation, NSDI 2013*, pp. 443–457, 2013.

[3] B. Balaji *et al.*, "Brick: Metadata schema for portable smart building applications," *Appl. Energy*, vol. 226, no. February, pp. 1273–1292, 2018, doi: 10.1016/j.apenergy.2018.02.091.

[4] P. Pauwels and G. Fierro, "A Reference Architecture for Data-Driven Smart Buildings Using Brick and LBD Ontologies," *CLIMA 2022 Conf.*, 2019, [Online]. Available: https://doi.org/10.34641/clima.2022.425.

[5] M. R. Bashir, A. Q. Gill, and G. Beydoun, *A Reference Architecture for IoT-Enabled Smart Buildings*, vol. 3, no. 6. Springer Nature Singapore, 2022.

[6] A. Krishnan Prakash *et al.*, "Solar+ Optimizer: A Model Predictive Control Optimization Platform for Grid Responsive Building Microgrids," *Energies*, vol. 13, no. 12, p. 3093, Jun. 2020, doi: 10.3390/en13123093.

[7] A. Bhattacharya, J. Ploennigs, and D. Culler, "Short Paper: Analyzing Metadata Schemas for Buildings," in *Proceedings of the 2nd ACM International Conference on Embedded Systems for Energy-Efficient Built Environments*, Nov. 2015, pp. 33–34, doi: 10.1145/2821650.2821669.

[8] B. Y. P. Fremantle, "Wso2_Whitepaper_a-Reference-Architecture-for-the-Internet-of-Things," vol. 0, 2015.

[9] Z. Chevallier, B. Finance, and B. C. Boulakia, "A reference architecture for smart building digital twin," *CEUR Workshop Proc.*, vol. 2615, pp. 1–12, 2020.

[10] M. Genkin and J. J. McArthur, "B-SMART: A Reference Architecture for Autonomic Smart Buildings.," *IOP Conf. Ser. Earth Environ. Sci.*, vol. 1101, no. 9, p. 092036, Nov. 2022, doi: 10.1088/1755-1315/1101/9/092036.

[11] M. Pau, P. Kapsalis, Z. Pan, G. Korbakis, D. Pellegrino, and A. Monti, "MATRYCS-A Big Data Architecture for Advanced Services in the Building Domain," 2022, doi: 10.3390/en15072568.

[12] D. Miha Smolnikar, David Carro and F. N. Pellegrino, Leandro Lombardo, "D2 . 2 | MATRYCS Technical and Security Specification," 2021. [Online]. Available: https://www.matrycs.eu/sites/default/files/2021-06/MATRYCS D2.2 - Technical %26 Security Specification_v1.0 %282%29.pdf.

[13] M. Pau, P. Kapsalis, Z. Pan, G. Korbakis, D. Pellegrino, and A. Monti, "MATRYCS—A Big Data Architecture for Advanced Services in the Building Domain," *Energies*, vol. 15, no. 7, pp. 1–22, 2022, doi: 10.3390/en15072568.

[14] INTRA, "BOOST 4.0 Reference Architecture Specification," no. 2019, 2019, [Online]. Available: https://boost40.eu/wp-content/uploads/2020/11/D2.5.pdf.

[15] T. R. Gruber, "A Translation Approach to Portable Ontology Specifications," *Knowl. Aquis.*, vol. 5, no. 2, 1993.

[16] G. Fierro *et al.*, "Interactive Metadata Integration with Brick," *BuildSys 2020 - Proc. 7th ACM Int. Conf. Syst. Energy-Efficient Build. Cities, Transp.*, pp. 344–345, 2020, doi: 10.1145/3408308.3431125.

[17] P. Pauwels, A. Costin, and M. H. Rasmussen, "Knowledge Graphs and Linked Data for the Built Environment," in *Structural Integrity*, vol. 20, 2022, pp. 157–183.

[18] L. Chamari, E. Petrova, and P. Pauwels, "A web-based approach to BMS, BIM and IoT integration: a case study," in *REHVA 14th HVAC World Congress*, 2022, pp. 1–8, doi: https://doi.org/10.34641/clima.2022.228.