IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 7 September 2023, accepted 9 October 2023, date of publication 19 October 2023, date of current version 27 October 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3325767

RESEARCH ARTICLE

An End-to-End Implementation of a Service-Oriented Architecture for Data-Driven Smart Buildings

LASITHA CHAMARI, EKATERINA PETROVA, AND PIETER PAUWELS
Department of the Built Environment, Eindhoven University of Technology, 5612 AZ Eindhoven, The Netherlands

Corresponding author: Lasitha Chamari (l.c.rathnayaka.mudiyanselage@tue.nl)

This work was supported by the Dutch Ministry of Economic Affairs and Climate Policy and the Ministry of the Interior and Kingdom Relations under the MOOI Program through the Brains for Buildings Project.

ABSTRACT Buildings connect with multiple information systems like Building Management Systems (BMS), Energy Management Systems (EMS), IoT devices, Building Information Models (BIM), the electricity grid, weather services, etc. Data-driven smart building software demands seamless integration of the above systems and their data. The lack of a system architecture with well-defined Application Programming Interfaces (APIs) poses a significant challenge for developing reusable, modular and scalable applications. This article presents a service-oriented system architecture designed with data-driven smart buildings in mind. The architecture relies on the Zachman framework and consists of seven service categories: 1) existing business applications, 2) new microservice-based applications, 3) databases, 4) integration software, 5) infrastructure services, 6) shared services, and 7) user interfaces. It closely resembles the MACH architectural principles: Microservices, API-first, Cloud-based components, and Headless principles. This architecture is implemented as a proof-of-concept, including three smart building applications. These include a Digital Twin application integrating sensor data with a BIM model, a web application merging real-time sensor data with semantic building graphs, and a data exploration tool using sensor data, the Brick ontology, and Grafana dashboards. Future implementations include real-time control applications such as Model Predictive Control (MPC). The proposed architecture and its implementations provide a blueprint for a reusable, modular, and scalable architecture in the smart building domain.

INDEX TERMS Building Information Modeling, data-driven buildings, data integration, IoT, linked building data, microservices, reference architecture, REST API, real-time data acquisition, solution architecture.

I. INTRODUCTION

Smart buildings must fulfil various operational objectives related to energy efficiency, energy flexibility, Indoor Environmental Quality (IEQ), occupant comfort and well-being, etc. Current approaches aiding the fulfilment of these objectives often rely on software programs designed for the purpose. The latter are commonly referred to as smart building applications [1], [2], [3] and tend to be data-driven. Examples of such applications include Demand Side Management (DSM) using flexible energy sources [4], [5], automated Fault Detection and Diagnosis (FDD) [6], [7], optimising Heating, Ventilation and Air Conditioning (HVAC) loads to achieve cost and comfort objectives [5], [8], [9], Digital Twins [10], [11], [12], etc. Data-driven applications rely on various components related to training machine learning models, forecasting energy demand, solving optimisation problems, and running predictive controllers such as Model Predictive Control (MPC). These applications also require supporting services such as data cleaning, aggregation, extract-transform-load (ETL) tools to gather and process data from different sources, etc. Since the above data are heterogeneous, integration is required and essential. Furthermore, system integration is also needed because the

The associate editor coordinating the review of this manuscript and approving it for publication was Fabrizio Messina.

systems that produce this data are diverse and use different technologies. On top of these back-end services are also the various dashboards and web applications used to visualise and convey useful information to the end user. When it comes to novel data-driven smart building applications, buildings cannot rely only on the limited capabilities of traditional Building Management Systems (BMS).

Smart building applications rely on operational and contextual data from various sources and systems. Operational time series data constitutes data from IEQ indicators, valve positions, faults, alarms, event logs, etc. These originate from systems such as BMS, EMS, weather services, etc., and are the primary input for any data-driven application. These data are mostly structured, and the technologies for handling this type of data such as time series databases, real-time data streaming engines, and big data frameworks, are mature and straightforward to implement.

On the other hand, Building Information Models (BIM), technical drawings, Process and Instrumentation Diagrams (P&ID), spreadsheets, and manuals contain a lot of information about the configurations and relationships between different systems and their components in a building. These constitute the contextual building data (metadata). Compared to operational data, these data are mostly unstructured. There is a growing interest in using semantic technologies such as ontologies to effectively manage and utilise these contextual building data in the architecture engineering and construction industry [13]. Despite their benefits, the technologies to harness and integrate this contextual data with other data are not equally mature.

Extracting, labelling, organizing, sharing and integrating both types of data for various applications is not straightforward [14]. Furthermore, many applications are developed and deployed as standalone services without allowing them to be reused or linked with other applications. The latter is mainly due to the absence of a well-designed software architecture and well-defined Application Programming Interfaces (APIs) to access systems, data and services. That has led to the use of ad-hoc procedures and tools when developing data-driven applications [15], [16], which have limitations when it comes to reusability, modularity, and scalability [16]. Therefore, smart buildings need to rely on a well-defined system architecture and APIs that enable both the integration of diverse systems and datasets and the orchestration of various services.

Each building is unique, given the diversity of the building stock, system vendors, and technologies. Therefore, developing a system architecture for each individual building is impossible. Here, an architectural pattern can be useful to represent a generic system architecture for a given building. These common patterns are also defined as a reference architecture, a blueprint or an architecture pattern. These patterns can support the creation of suitable instances of system architectures for a given domain by introducing reusable knowledge and best practices.

Examples of such patterns are already widely available in the software architecture domain. For instance, Fernando [17] presents such patterns for the transportation, telecommunication, retail, education, automotive, healthcare, power & energy and hospitality industries. Although some features of these architectures are common for smart buildings, they cannot be directly transferred to the smart building domain. There are few established architectural patterns available for smart buildings in the literature. Most existing ones are either too abstract, largely conceptual, do not follow well-established architecture frameworks or patterns, not vendor-agnostic or a combination of the above. The absence of such architecture hinders the development of reusable, modular and scalable applications for buildings. In turn, the achievement of the operational objectives remains challenging.

This article presents an architecture for smart buildings aiming at developing data-driven applications in a reusable, modular and scalable manner. Two major focus areas are data integration and system integration. These are the critical yet less investigated aspects of smart building architecture. Data and systems are also unique for the building ecosystem compared to other domains such as transportation, telecommunication, etc. The design of the proposed architecture relies on three main inputs:

1) Understanding the smart building system architecture with the help of the Zachman architecture framework [18].
2) Deriving functional and non-functional requirements based on interviews with building stakeholders, a literature review on smart building applications, and lessons learned from existing smart building application development efforts [19].
3) Investigating already available, well-tested architecture patterns in the Information Technology (IT) industry [17], [20].

Based on the above, the article presents a proof-of-concept implementation of an instance of the proposed architecture in three smart building applications. As such, the objectives, and therefore, the novel contribution of this paper is three-fold:

Objective 1: Develop an architecture for data-driven buildings informed by the Zachman architecture framework, the state-of-the-art in the scientific domain and existing architecture patterns. This architecture satisfies the identified functional and non-functional requirements of a data-driven building. The proposed architecture also aligns with the MACH architectural principles [21] and relies on *a) Microservices; b) Application Programming Interfaces (API)-first (exposing functionality via APIs), c) Cloud-based components, and d) Headless principles (front-end and back-end logic are decoupled).*

The artefacts of the architecture are logically grouped into seven categories for better understandability and separation of concerns. The seven categories are 1) existing business applications, 2) new microservice-based applications,

3) databases, 4) integration software, 5) infrastructure services, 6) shared services, and 7) user interfaces. The system architecture is further developed based on previous results from integrating BIM, BMS and IoT data on the web [22] and is continuously updated based on new use cases in smart buildings.

Objective 2: Demonstrate how to integrate semantic technologies with the smart building software architecture, thereby effectively reusing existing ontologies and semantic technologies for metadata integration.

Objective 3: Implement an instance of the architecture and demonstrate the implementation in three use cases, thus providing an *end-to-end solution*. These three applications include 1) a Digital Twin application integrating sensor data with a BIM model, 2) a web application integrating real-time sensor data with semantic building graphs, and 3) a data exploratory tool using sensor data, the Brick ontology and Grafana dashboards.

The remainder of this article is organised as follows. Section II discusses the background of software architectures and the related work in software architecture development for smart buildings. Section III presents the research method relying on six steps towards designing and implementing the proposed architecture. Section IV highlights the results relative to the six methodological steps and includes the resulting architecture, implementation and use cases. Finally, Section V concludes the article by discussing the practical implications and future work.

II. BACKGROUND AND RELATED WORKS

This section investigates the background, terminology and related works in the software architecture domain. The study then investigates existing software architectures for smart buildings. We discuss how the available software architectures are applied to the smart building domain. The section concludes with the identified research gaps and the main contributions of the paper.

A. SOFTWARE ARCHITECTURE, ENTERPRISE ARCHITECTURE, ARCHITECTURE PATTERNS, REFERENCE ARCHITECTURE AND SOLUTION ARCHITECTURE

The definition of architecture in this context is ''the set of principal design decisions about the system'' [23]. Similarly to how architectural principles in the built environment help develop complex buildings, software architecture plays a crucial role in developing complex software systems. It provides a structured approach to designing, planning, and organising the various components and modules of the software application. The architecture is the backbone of a successful software system [24]. When applied to a broader enterprise scope, it is referred to as enterprise architecture, which captures both business stakeholders and technical IT aspects.

Software architectures that are useful and applicable to specific domains or systems are recognised as architectural patterns and reference architectures that provide the blueprint for creating a suitable architectural solution [23], [25], [26]. Fernando [25] defines the implementable version of an architecture as a ''solution architecture'', which also has three components: business, technical, and deployment. Fig. 1 shows the relationship between an enterprise and a solution architecture. As such, one enterprise architecture enables many different solution architectures (n) for different use cases. According to TOGAF [27], a full-scale enterprise architecture consists of four types of architectures: business architecture, data architecture, application architecture and technology architecture, which are considered subsets of the overall enterprise architecture. These subsets also become different viewpoints for the different stakeholders. In practice, stakeholders (e.g., building owners, engineers) have different interests, and the different architectural views help define access rights and reduce inefficiencies and errors [28].

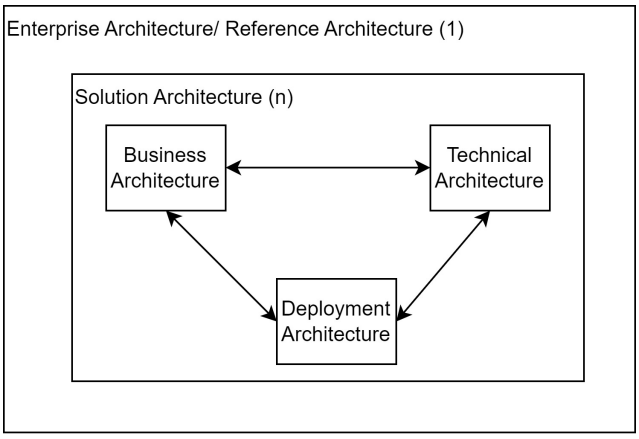

FIGURE 1. Relationship between enterprise architecture and solution architecture (Fernando [25]). One reference architecture can be the basis for many solution architectures.

B. ARCHITECTURE FRAMEWORKS

There are various architecture frameworks that guide the design of a software architecture. ISO/IEC/IEEE 42010 defines an architecture framework as *''conventions, principles and practices for the description of architectures established within a specific domain of application and/or community of stakeholders''*. The two most commonly used frameworks are the Zachman framework [29] and The Open Group Architecture Framework (TOGAF) [27].

The Zachman framework, introduced in 1980, provides the taxonomy for modelling an enterprise. It helps arrange the requirements and the reasoning behind them from 36 different perspectives from two dimensions, as shown in Table 1. The Zachman framework has been used in several academic and industrial settings, including [30], [31], [32], [33], and [34]. This framework provides a clear image of the needs and responsibilities of all stakeholders involved in the development of an enterprise architecture. Nevertheless, it does not provide a method for creating an architecture.

TOGAF is another industry standard that assists in accepting, producing, using, and maintaining enterprise architectures. It provides a set of definitions, a process for creating

TABLE 1. Zachman architecture framework [29].

Perspective	Data (What)	Function (How)	Network (Where)	People (Who)	Time (When)	Motivation (Why)
Scope (Planner)	List of things important to the business	List of processes the business performs	List of locations in which the business operates	List of organisations important to the business	List of events significant to the business	List of business goals/strategies
Business Model (Owner)	Semantic model	Business process model	Business logistics system	Work flow model	Master schedule	Business plan
System Model (Designer)	Logical data model	Application architecture	Distributed system architecture	Human interface architecture	Processing structure	Business rule model
Technology Model (Builder)	Physical data model	System design	Technology architecture	Presentation architecture	Control structure	Rule design
Detailed Representations (Sub-contractor)	Data definition	Program	Network architecture	Security architecture	Timing definition	Rule specification
Functioning Enterprise	Data	Function	Network	Organisation	Schedule	Strategy

an enterprise architecture, and tooling to support the development [35]. Examples of using TOGAF as a reference are demonstrated in [35], [36], [37], and [38].

C. ARCHITECTURE DESCRIPTION

Architecture Description (AD) serves as a means to express and communicate the system architecture [24]. There are several established modelling languages for this purpose. The Unified Modelling Language (UML) [39] is the industry standard for specifying, visualising, constructing, and documenting software system artefacts. Additionally, ArchiMate [40], published by The Open Group, is another notable language. These visual languages also provide a standardised set of icons that facilitate unambiguous communication of architectural concepts. Furthermore, various authors also employ custom icons and languages tailored to their specific needs. Hence, an architectural language can take many forms, ranging from informal box-and-line notations to more formal Architecture Description Languages (ADLs) [41]. Ultimately, the goal is to provide the necessary information for stakeholders to understand the architecture and enable effective decision-making [41], [42], [43].

D. CHOOSING AN ARCHITECTURE PATTERN

One way to look at architecture patterns is by domains. These patterns are available for different domains, e.g., automotive industry, healthcare, banking, etc., and result from the experience and best practices of software architects [17]. Patterns provide a starting point when designing a new architecture for a given use case. Leveraging or adapting an existing architecture to suit one's specific requirements offers numerous advantages. One key benefit is the significant reduction in time spent reinventing solutions that have already been thoroughly tested and proven effective in various enterprises. By doing so, one can utilise a well-tested and fail-safe architecture that incorporates best practices [17].

Another way to look at architecture patterns is according to their underlying software technology. As such, Richards [20] provides two major categories for architecture patterns:

TABLE 2. Comparison of architecture characteristics (adapted from Richards [20]). One dot means the architecture characteristic is not well supported, five means it is well-suited.

Aspect	Layered (Monolith)	Event-driven	Micro-services	Cloud-based
Overall Cost	●	●●●	●●●●●	●●●●●
Agility	●	●●●	●●●●●	●●
Simplicity	●●●●●	●	●	●
Scalability	●	●●●●●	●●●●●	●●●●●
Fault Tolerance	●	●●●●●	●●●●●	●●●
Performance	●●●	●●●●●	●●	●●●●●
Extensibility	●	●●●●●	●●●●●	●●●

monolithic (single deployment units) and distributed (multiple deployment units, usually consisting of services). Layered architecture (n-tier), microkernel architecture, and modular monolith are considered monolithic architecture patterns, whereas event-driven, microservices, service-oriented and space-based (cloud-based) architectures are considered distributed. Deciding which architectural pattern to use depends on how each pattern's strengths and weaknesses will impact the problem at hand [20]. This can be solved by analysing the common characteristics of these software architectures. Table 2 summarises the strengths and weaknesses of software architecture patterns as expressed by Richards [20].

The definition of the characteristics is as follows:

1) Overall cost: The cost for designing, developing and running a system architecture.
2) Agility: The capability of a system architecture to adapt to constantly changing requirements.
3) Simplicity: Avoiding unnecessary complexity by using fewer components, services and repositories.
4) Scalability: Ability to grow system capacity over time with the number of users or requests increase.
5) Fault tolerance: Ability of the system to continue functioning in an event of error in a component.
6) Performance: The amount of time it takes for the system to process a business request.
7) Extensibility: The ease in which a system can be extended with additional features and functionality.

Table 2 shows that some architecture patterns offer high scalability, but this comes with an increased overall cost. Other architecture patterns can be simple to develop but not scalable or extensible. Most importantly, no single architecture outperforms the rest, and understanding each pattern's characteristics, strengths, and weaknesses is necessary to choose the one that meets specific business needs and goals. The state-of-the-art in the area generally discusses monolith, service-oriented (SOA) and microservices-based ones [44], [45], [46], [47], [48], [49]. A monolithic architecture denotes the traditional method of creating a software program as a single, unified entity, functioning independently of other applications. One notable advantage of a monolithic architecture is faster development due to its straightforwardness and a single code base. Furthermore, monolithic architectures exhibit higher performance, outperforming, e.g., microservices, due to the absence of network communication overhead between separate services [49]. However, the drawbacks of a monolithic architecture also become apparent, particularly in terms of extensibility, agility, fault tolerance, and scalability.

In contrast to monolithic architecture, SOA is a modern architectural concept that emphasises loose coupling, reusability, interoperability, agility, and efficiency. The key principle of SOA involves breaking down each business process into smaller, modular blocks of tasks and functions known as services [50], [51], [52], [53]. To facilitate the integration of various services, SOA often employs an Enterprise Service Bus (ESB) [54]. ESB acts as a central hub, enabling seamless communication between different services and providing a standardised approach to message routing, transformation, and mediation.

The microservice architecture is the newest architecture pattern. Some studies also consider microservices as a subset or a successor of SOA [47], [55] with better performance. Fig. 2 illustrates the basic topology for the microservices architecture style. It consists of individual, specialised services deployed independently and accessed through an API gateway. As the application grows, microservices can be updated in an isolated manner and deployed without affecting the rest of the applications, scoring the highest in agility and extensibility in Table 2. In this architecture style, each service is responsible for managing its data. The communication between services is straightforward and efficient because of its request-response and event-based message styles [47]. In the microservices architecture, extending functionality is often a straightforward process that includes creating a new service, encapsulating it within a container, establishing an API endpoint, and deploying the service [20]. This makes horizontal and vertical scaling easier compared to monolith or SOA architecture, which explains the higher score in scalability and extensibility in Table 2. Due to the large number of microservices available in a complete solution, managing the testing, deployment, and monitoring of such a large number of services are done using containerisation technologies such as Docker and service orchestration and management platforms such as Kubernetes. These artefacts

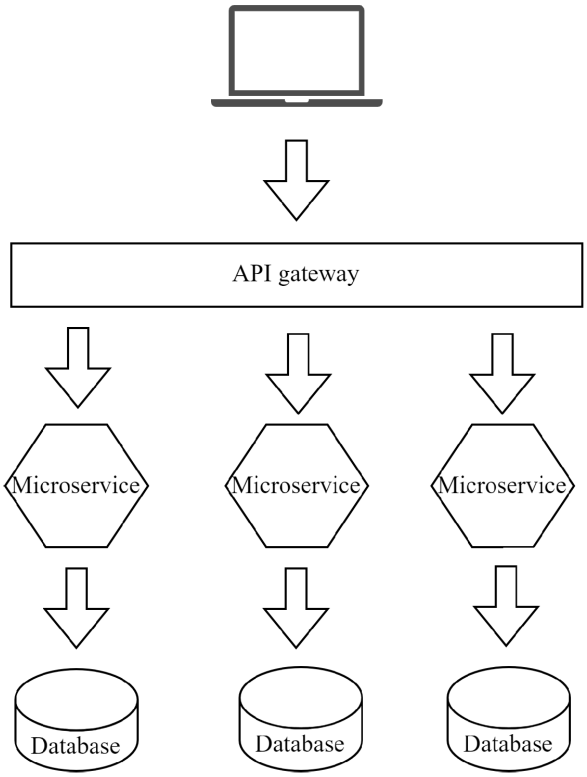

FIGURE 2. The basic topology of the microservices architecture (adapted from Richards [49]).

make the microservice architecture more agile and scalable but also reduce the simplicity and performance due to latency in the communication (network latency, security latency, and data latency) [49] between remote services as depicted in Table 2.

E. DATA-DRIVEN BUILDING DOMAIN

Traditionally, there were few software applications besides a BMS in a building. Therefore, the software architecture concept was not very prominent in the building domain. Nowadays, the focus on using data to support the growing objectives related to energy efficiency, energy flexibility, IEQ, occupant comfort, well-being, etc., leads to the rapid development of related software applications.

Smart building applications range from monitoring to data-driven control of built assets. Some cutting-edge applications include FDD in HVAC systems [56], [57], MPC for optimising energy usage in buildings [5], [9], [58], or Digital Twins which are virtual replicas of the physical assets in a building [10], [11], [12] providing ample opportunities for real-time monitoring, identifying and locating faults, etc.

The first requirement for data-driven applications is the ability to integrate various systems. This is mainly due to two reasons, i.e., to utilise their data and to establish live communication with the controllers that can execute the optimum results generated from these data-driven applications. Such systems may include BMS, EMS, IoT devices, BIM, and external services for weather and grid price signals. In this

regard, various tools aiming to acquire data from heterogeneous sources have been proposed in the literature. For example, Prakash et al. [5] and Zhang et al. [58] demonstrate the ingestion of time series data from on-premise devices, such as thermostats, battery storage, and IoT devices, using ''drivers'' that conform to specific device protocols such as Modbus or BACnet. These data are then published to a message queue. Applications communicate with this message queue to get the needed data. Optimum set points are also published to the message bus, and the drivers receive the set points to actuate the necessary changes. Agarwal et al. [15] utilise ''data connectors'' - programs that retrieve data from the sensor network and write data back for actuation. This approach enables bi-directional communication, which is essential for data-driven building controls. Based on the literature, protocol adoption is essential to data ingestion since no single protocol dominates the implementations. Related works in [1], [5], [59], [60], [61], and [62] employ different concepts such as middleware, proxies, or adaptors. Yet, the concept remains the same and follows a similar logic.

The second requirement is the data integration. With a lot of operational and contextual data coming from various systems, the ability to link and integrate them with applications is crucial. To that end, a large body of recent research explores the creation and use of domain ontologies. An ontology is a structured and formalised model representing the knowledge within a given domain [63]. It involves formalising concepts, relationships, classes, and attributes within the domain. In an ontology, concepts are represented as nodes in a graph data structure, while the edges represent the relationships between them. Ontologies in the building domain such as the Building Topology Ontology (BOT) [64], [65], Semantic Sensor Networks (SSN) [66], Sensor-Observation-Sampling-Actuation (SOSA) [67], Building Automation and Control Systems (BACS) [68], Brick [2], [69] and Haystack [70] are used to instantiate a standard semantic description of a given system such as BMS, BIM and IoT. These ontologies not only standardise the semantics but also enable linking heterogeneous domains. The use of ontologies in the data-driven building context has been demonstrated in a variety of use cases, including BIM and IoT integration [22], context-aware control of mechanical systems [71], [72], automating KPI calculation for building performance [73], automatic setup of FDD for BMS [74], etc.

F. SOFTWARE ARCHITECTURES FOR THE SMART BUILDING DOMAIN

Multiple reference architectures are already available in the literature for different domains. For instance, the Industrial Internet Consortium (IIC) presents the Industrial Internet Reference Architecture (IIRA) [26] for the IoT domain. They identify five disciplines of a typical IIoT (Industrial IoT): control, information, operations, application and business domains. They map the identified disciplines into a three-tier architecture with an edge, platform and enterprise tier. IIC also provides four architecture viewpoints, namely business, usage, functional and implementation viewpoints.

The AIOTI reference model [75] is another high-level IoT architecture with three layers, including network, IoT and application layers. Compared to IIRA, AIOTI provides less use of architecture frameworks and demonstrates a very high-level representation of the IoT domain. However, IIRA and AIOTI are reference architectures and do not propose any solution architecture for the data-driven building domain.

The BDV Reference Model [76] serves as a common reference framework around big data technologies on the overall IT stack. Here, the primary source of big data is sensor data from an IoT context and actuator interaction in cyber-physical systems. Again, this is a generic high-level reference architecture that does not provide a solution architecture.

Within the scope of big data, another existing architecture in the smart buildings context is the MATRYCS [16]. MATRYCS is a reference architecture for building services utilising six design principles: modularity via microservices, cloud virtualisation, openness and data sharing, security, no vendor lock-in, and distributed data ecosystem. This high-level architecture consists of four layers: i) infrastructure, ii) governance, iii) processing and iv) analytics. These layers encompass services such as data ingestion, interoperability, data processing, streaming, data harmonisation, querying, reasoning and sharing. The authors also propose a solution architecture; however, a demonstration of the applicability of the proposed architecture in a real-world use case is missing.

Mazzara et al. [77] propose an ICT reference architecture for smart buildings containing four layers: hardware, network, management and application & service. This is also a high-level architecture, and a specific solution architecture is not presented.

The BuildingDepot architecture [15] focuses on enabling the implementation of portable data-driven applications on top of the distributed physical resources present in large commercial buildings. Its main principles are scalable data storage, ease of data access, fine-grained data sharing, and access control. The architecture relies on REST APIs for system interaction, as demonstrated in the presented use cases. However, the architecture does not include other essential aspects such as metadata management, integration of external services, or user interfaces.

Chevallier et al. [10] proposes an architecture focusing only on the integration of sensor data with smart building ontologies. However, this architecture only describes a part of the smart building ecosystem. Abu-Matar [78] proposes a data-driven reference architecture for smart cities inspired by SOA. However, the result is limited to a UML meta-model for the views, their corresponding elements and relationships. These custom views create ambiguity when expressing the different artefacts in the architecture, instead of utilising existing architecture frameworks such as Zachman or TOGAF.

Genkin and McArthur [79] introduce B-SMART, a reference architecture that facilitates the autonomic optimisation of smart buildings. B-SMART is also a high-level conceptual layered architecture, the implementation of which is yet to be seen.

Bashir et al. [80] proposed the IBDMA reference architecture, which also uses standard drivers to ingest data; however, it does not include any insight related to data integration or bi-directional communication. Bashir et al. [81] also provides a metamodel, which can be considered a taxonomy of the concepts around big data and IoT. However, this metamodel does not rely on existing architecture frameworks or semantic technologies, which can be a limiting factor in terms of extensibility. Creating new data models instead of reusing and extending existing ones limits the reusability of the applications.

Bao et al. [59] propose a microservice architecture for IoT and the energy domain in smart buildings. Their architecture includes heterogeneous systems, drivers for protocol adaption, databases, and data-driven applications. They utilised a message hub to establish communication between the above services. However, this architecture is also presented at an abstract level and does not have an implementation in real-world use cases.

There are also reference architectures provided by commercial parties, which showcase the different products a service provider can offer to develop an architecture for a data-driven building. Microsoft Azure provides reference architectures for various applications. The closest category to smart buildings is the IoT category, which provides 70 different architectures [82]. Amazon Web Services (AWS) also provides 48 reference architectures [83] under the IoT category. Due to the fully cloud-based nature of these architectures, they inherently support characteristics like scalability and fault tolerance. However, these architectures still need the domain-specific components identified from the smart building domain, such as drivers for bi-directional communication, smart building ontologies, etc. Therefore, they cannot be a direct reference for a data-driven building.

In reality, various systems and services in buildings are individually installed by different vendors at different times, and there is no overarching architecture. This fact is not well-represented in any smart building architectures proposed in the literature. When adopting architecture patterns from the software architecture domain to a building, what exists in a building can be closely represented by the brown-field enterprise architecture illustrated by Fernando [25] in Fig. 3. This kind of mixed enterprise environment (i.e., brown-field or a blended architecture) [84] is a combination of components from different architecture patterns [25] utilised to realise business needs. In fact, according to Fernando [25], what is mainly observed in many enterprises is a brown-field architecture rather than a clean architecture (i.e., based entirely on microservices). It is possible to design a new architecture with microservice principles in mind (Modern Application Layer in Fig. 3). However, the existing heterogeneous applications (top layer in Fig. 3) also need to coexist with the new architecture. A software architecture aims to integrate existing legacy systems while allowing modern applications to be integrated following the best practices in software architecture patterns.

G. RESEARCH GAP AND CONTRIBUTION

Based on the performed literature review, the following three research gaps have been identified:

1) The existing architectures for smart buildings generally do not consider best practices (e.g., architecture frameworks) and existing architecture patterns. Achieving implementable solutions requires an adaptation of the best practices for the smart building domain, as stated in Objective 1.
2) Smart building research has demonstrated the value of semantic technologies to data integration in smart buildings. However, previous studies have not integrated such technologies as an architectural component and exhibit limited capabilities using ad-hoc tools. Metadata integration plays a crucial role in the architecture and should be implemented in the smart building architecture (Objective 2).
3) Most reference architectures are presented at abstract levels, and the technical details required for implementation are scarce. Therefore, it is important to demonstrate the proposed system architectures with an *end-to-end solution* for better understandability and usability. This study proposes a solution architecture and provides example implementations in three smart building use cases, as highlighted in Objective 3.

III. RESEARCH METHOD

This section discusses the six-step research method associated with designing and implementing the architecture for smart buildings.

Step 1: The first step is to understand the data-driven control domain of an existing building ecosystem in terms of the requirements coming from key stakeholders. This step builds on previous work investigating the data needs and requirements for smart buildings [85]. The latter were identified based on a series of interviews with relevant stakeholders (building owners and technical solution providers).

This study relies on the Zachman architecture framework to organise the above information as a guideline for designing a system architecture. Every cell containing the relevant artefacts helps achieve a comprehensive understanding of the system from each stakeholder's viewpoint. For example, ''data'' for the business owner is the essential information about the business, while it means physical data models for the database administrator. Using the Zachman framework helps establish a clear link between the business requirements and their corresponding technical implementations.

Step 2 : The next step defines the functional and non-functional requirements of the system. These requirements are informed by the previously performed interviews

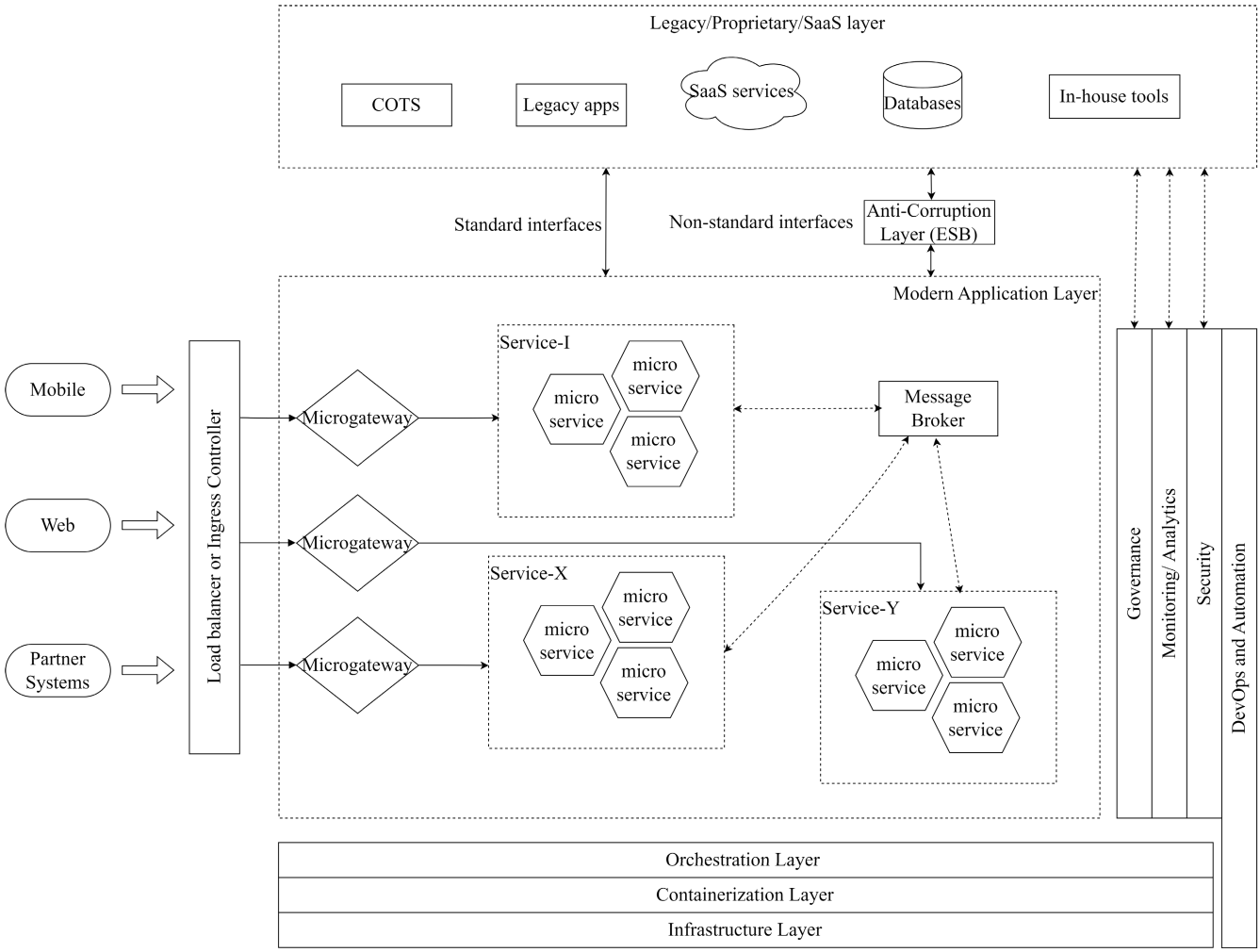

FIGURE 3. Microservices based architecture for a brown-field enterprise where a combination of existing and new applications co-exists to perform a business task [25]. Image reproduced with the permission from the original author.

and the state-of-the-art review. Defining a set of functional and non-functional requirements helps determine the architecture pattern and identify the necessary software components of the architecture.

Step 3 : This step entails the selection of an architecture pattern suitable for the smart building ecosystem identified in Step 1 and the requirements specified in Step 2. This decision is strongly influenced by the nature of the smart building ecosystem, which is identified as brown-field in the literature. A smart building already has multiple software systems, and modern applications for data-driven control should be implemented as independently deployed services by connecting with existing software systems.

Step 4 : The next step defines the components of the architecture. This is achieved by i) identifying existing systems in buildings, ii) identifying common components by studying reference architectures and architecture patterns available in other domains, and iii) determining which new or unique components will be needed for the smart building domain based on the requirements from Step 2 and the architecture pattern from Step 3.

Step 5 : This step concerns the development of the architecture. Here, we propose a solution architecture, mainly focusing on the technical architecture. At this stage, the definition of the system architecture for a smart building is complete.

Step 6 : The last step includes the implementation of an instance of the proposed deployment architecture, which is an additional step we take to provide a proof-of-concept for different applications. As seen in the literature review, designing and implementing an architecture is a non-trivial process. It involves several stakeholders, software architects, developers and deployment infrastructure. In this article, we focus on the technical architecture and demonstrate the implementation in three smart building applications.

IV. RESULTS

A. STEP 1 : DATA-DRIVEN BUILDING ECOSYSTEM IN THE ZACHMAN FRAMEWORK

Table 3 illustrates how the data-driven smart building ecosystem is mapped to the Zachman framework. Here, specific artefacts are described with different purposes for different

roles. Since Zachman framework is an enterprise architecture design framework, this mapping takes into account six viewpoints from six different stakeholders. All three components of the solution architecture identified from the literature (business, technical and deployment) are placed in their appropriate cell and enclosed in a box. Since the main focus of this study is the technical architecture, artefacts that are important for the technical architecture are represented in Bold.

B. STEP 2 : FUNCTIONAL AND NON-FUNCTIONAL REQUIREMENTS

In the following section, we present the functional and non-functional requirements of the system architecture. Functional requirements describe what the system should do to support the user actions, while non-functional requirements describe how the system should operate.

1) FUNCTIONAL REQUIREMENTS

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 several existing devices, services, and applications.
6) FR6 Real-time bidirectional communication ability to automatically or autonomously control facilities within the smart building.
7) FR7 User-centred interfaces for easy information access.

2) NON-FUNCTIONAL REQUIREMENTS

Most non-functional requirements are common to a system architecture regardless of domain or architecture pattern. These include performance, scalability, extensibility, security, reliability, and maintainability. In addition, the smart building context also demands the following non-functional requirements:

1) NFR1 Reusability of components and services implemented in the architecture for a wide range of smart building applications.
2) NFR2 Add new services in a modular way. Each module performs a specific and well-defined function, and these modules can be developed, tested, and maintained independently.
3) NFR3 Scalability to manage the increasing amount of devices, data or services in the building.

C. STEP 3 : ARCHITECTURE PATTERN

The aim is to define an architecture for a data-driven smart building, building upon existing architecture patterns closely resembling the data-driven building ecosystem. The literature review identified that the brown-field architecture presented in Fig. 3 is a good starting point. There is a requirement to support seamless access to various existing sources and their data (FR1, 2, 3), and their integration (FR4). Included here are the new services (NFR2) to be deployed such as data-driven applications. Multiple services are already available in a building; the goal is to keep and integrate them with new services (NFR2). Based on the literature review, the microservice architecture is more suitable for new applications because its modular approach facilitates scalability, reusability and extensibility (NFR1, 2, 3). Furthermore, there is a need to transport commands and messages between different services (FR5, 6), and microservice-based architecture frameworks inherently support this by providing the necessary transport mechanisms. This inter-service message transport is essential in sophisticated control strategies such as MPC to trigger one service (e.g. calculate optimum set points) based on the output of another service (e.g. building load forecast). This type of coordination is usually achieved by using ''event-based messaging'' in microservices. Further, in terms of adding new services in a modular way (NFR2), as previously mentioned, extending the functionality of a microservice architecture is often a straightforward process involving creating this service, encapsulating it within a container, establishing necessary API endpoint, and deploying it [20].

D. STEP 4 : IDENTIFYING SOFTWARE COMPONENTS

This section presents the main artefacts that will constitute the architecture. Some components are derived from Fig. 3 based on the literature review, and the others are specific to the data-driven building domain. The list of components is as follows:

1) Existing business applications
2) New microservices-based business applications
3) Databases
4) Integration software
5) Infrastructure services
6) Shared services
7) User interfaces

1) EXISTING BUSINESS APPLICATIONS

Existing business applications can be grouped into three categories: 1) In-house devices and systems, which includes the BMS, EMS, IoT devices, etc., 2) external services such as weather APIs, utility services, e-mobility platforms, IoT platforms, etc., and 3) other metadata sources such as P&ID diagrams, BIM models, installation specifications, etc. The integration of these services with the overall system architecture is discussed in the integration software section.

2) NEW MICROSERVICES-BASED APPLICATIONS

As mentioned, the architecture will rely on the microservice pattern when introducing new applications. We identified three types of new applications: 1) drivers/connectors for

TABLE 3. Smart data-driven building ecosystem in the Zachman Framework. Different components of the enterprise architecture are places in suitable cell and highlighted in Blue. Components in Bold are elaborated in the upcoming sections.

Perspective	Data (What)	Function (How)	Network (Where)	People (Who)	Time (When)	Motivation (Why)
Scope (Planner)	**Business architecture** business objectives, strategic goals, high-level requirements for smart building	Strategies for achieving energy efficiency, occupant comfort, and other objectives	Geographical locations of buildings and facilities requiring smart technology	Key stakeholders, decision-makers, facility managers, occupants and beneficiaries	Timelines for implementing architecture, smart technologies, and achieving specific objectives	Justification for adopting smart building technology such as compliance to legislation, occupant satisfaction and well-being, and alignment with business goals
Business Model (Owner)	Business processes related to facility management, maintenance, and operations	Methods and procedures for managing building systems and addressing maintenance issues	Locations where facility management activities are conducted	Roles and responsibilities of facility managers, maintenance staff, third party service providers, and occupants	Timing of regular maintenance tasks, emergency responses, and system updates	Cost savings, benefits to the organisation, and occupant satisfaction.
System Model (Designer)	**Functional and non-functional requirements of the system, data processing requirements, logical data models, real-time monitoring needs,** real-time control needs	**Technical architecture**	**Deployment architecture** Locations where system components are deployed (on-premises, cloud, edge)	Development teams, system administrators, technical support personnel	Development milestones, system deployment schedules, software updates	Rationalisation for specific technology choices, system performance objectives, and operational efficiency
Technology Model (Builder)	**Software modules, integration interfaces, APIs, hardware components**	Software design patterns, software development methodologies	Physical locations of servers, sensors, and devices where components are deployed	Development teams, integration experts, hardware suppliers	Component development and integration timelines, system maintenance schedules	Component reusability, modularity and scalability, system reliability.
Detailed Representations (Sub-contractor)	**Network infrastructure, IoT platforms, building automation network, data storage and processing technologies**	Network design, protocols, security measures, data encryption methods	Physical and virtual locations of servers, gateways, and data centres	Technology vendors, partners, service providers	Timelines for developing, software integration, and technology upgrades	Justification for technology selection
Functioning Enterprise	**User interfaces, dashboards, reports, control mechanisms**	User experience design, interaction patterns, user training methods	Locations where users access interfaces (desktop, mobile, dashboards)	Facility managers, maintenance personnel, occupants	Operational hours, user activity patterns, maintenance windows	Improved user satisfaction, ease of operation, real-time insights, remote monitoring capability

new systems (e.g., IoT drivers, BMS drivers, metadata schema generators), 2) data-driven applications themselves (e.g., MPC, FDD) and 3) supporting services for data-driven applications (e.g., data aggregation services, data cleaning services, and ETL processes). This ability to integrate different services when needed is the key to extending the possible applications that can be implemented atop the system architecture without disrupting the existing ones.

3) DATABASES

The proposed system architecture needs to facilitate different types of data, such as time series data,semantic graphs (usually based on the Resource Description Framework (RDF)), object data (such as Industry Foundation Classes (IFC) files, geometry files, PDFs), and other data, such as user and project data. These constitutes structured (e.g., CSV, spreadsheets), semi-structured (e.g., JSON, XML) and unstructured (e.g., PDF, images) data. According to the data type, several

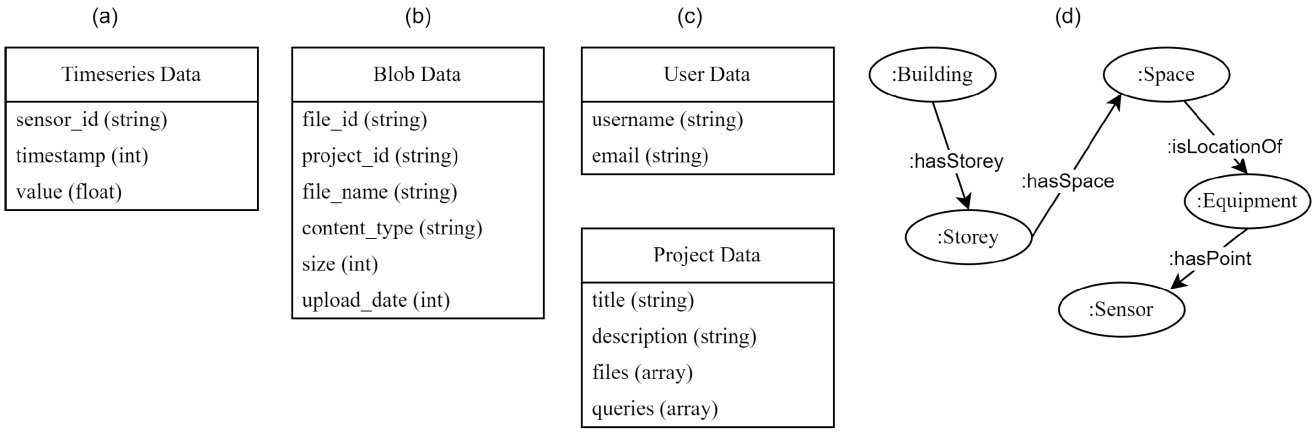

FIGURE 4. Logical data models of a) time series database b) Blob storage c) Document database and d) Graph database.

databases need to be selected to make data storage and querying more efficient. Time series databases are optimised to store and query time series data, whereas object databases are suitable for storing files. Object storage can also be used to store archived large files and sensor data archives, which can be used for data-hungry applications relying on Machine Learning. User and project details of different applications also need to be saved, ideally in a relational or document database. Semantic graphs of building metadata are stored in a graph database. As such, the proposed architecture relies on four types of databases:

1) Time series database
2) Graph database
3) Blob storage
4) Document database

The logical data models for these four databases are shown in Fig. 4. A logical data model entails the conceptual representation of the data without the specific technical details of the database management system.

4) INTEGRATION SOFTWARE

Integration software allows applications of different vendors that use different protocols to extract and translate their data into desirable formats and communicate them to relevant services. In a microservice–based platform, integration software can also be recognised as microservices. Depending on the three existing business application categories mentioned above, various drivers need to be in place. Then, standard interfaces such as HTTP and message queue systems such as Kafka, RabbitMQ, or NATS [86] can be used for the integration with the architecture [55]. Five integration software types essential for the data-driven building domain are identified below.

a: BMS INTEGRATION SOFTWARE

The BMS driver is the interface between the BMS and the system architecture. An example scenario is the communication of the data from a BACnet device to a web service. This would require a communication gateway (driver) capable of translating BACnet data into a format suitable for web communication (like JSON) and then communicating them via HTTP to a remote web service endpoint. Access to the data and control of the devices of the BMS is essential since many controllers are connected to the BMS. Usually, the sensor data are periodically downloaded to a database to use as historical data. Furthermore, a live communication link with the controllers needs to be established to send control commands to them based on the optimum results of data-driven building applications.

b: IoT INTEGRATION SOFTWARE

Although the BMS is typically the main source of sensor data, many buildings implement IoT devices as a retrofit. They provide access to streams of real-time sensor data. For any given building, many such devices and IoT platforms are available and need to be integrated before utilising their data for an application. Therefore, IoT integration software is important, such as the one proposed earlier by the authors [87], where sensor data acquired from different IoT sources (Node 1 of Fig. 5) are published into a central message broker (Node 2 of Fig. 5), and then processed to a uniform format (Node 3 of Fig. 5). Here, the real-time data can be accessed through a message broker, such as Mosquitto, using the MQTT protocol, a lightweight, publish-subscribe network protocol that transports messages between devices. MQTT follows a "topic" and "payload" method to communicate data. Under one topic, it is possible to communicate multiple data points. Listing 1 shows the logical data model of the outgoing IoT payload. This IoT data can also be recorded in a time series database for later usage.

```
{
  topic: 'device_model/mac_address/pub',
  keys: ['field-1', 'field-2','field-2'],
  values: ['value-1','value-2','value-3'],
  timestamp: unix timestamp
}
```

Listing 1. Logical data model of the uniform IoT payload.

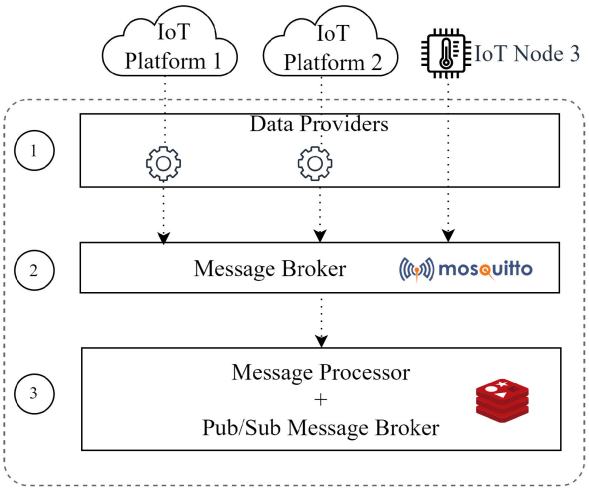

FIGURE 5. IoT integration service to acquire and unify payloads originating from heterogeneous sources [87].

c: BIM INTEGRATION SOFTWARE

BIM models are usually developed using commercial software like Revit. These models consist of architecture, structural and Mechanical Electrical and Plumbing (MEP) models from the design stage. When integrated into the operational stage of the building, they can bring huge benefits such as creating Digital Twins of an operational building. To integrate these models, one can rely on commercial off-the-shelf products like Autodesk Tandem for creating Digital Twins. The downside, however, is the vendor-lock-in and less flexibility and limited extensibility because of their closed APIs. Here, we propose an alternative approach using emerging openBIM software. Here, the IFC model, a vendor-neutral format of BIM used to exchange information, can be used. Standard libraries such as IFC.js [88] or xeokit [89] then allow creating front-end web applications to interact with the IFC models, ultimately enabling interaction with the BIM model via a web browser. Here, the Global Unique Identifiers (GUID) of elements in the BIM model can be used to link the BIM objects with their operational data, which enables the creation of Digital Twins.

d: METADATA INTEGRATION SOFTWARE

We use smart building ontologies to create an extensible semantic graph (also called a metadata schema) containing the contextual information about the building and its data. The graph carries the physical, logical and virtual assets in buildings and the relationships between them. Furthermore, it contains references to external systems such as time series databases and IFC models, enabling the automatic querying and retrieval of time series data associated with the instances in the metadata schema.

Generating this kind of metadata schema requires converting structured or unstructured metadata from various sources into subject-predicate-object triples that conform to the desired ontology. Conversion of the available metadata in different systems like BMS and BIM from different formats like CSV or IFC into a metadata schema requires a suitable driver. Here, we show two essential metadata integration software.

1. IFC to RDF driver: 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. The converter in [90] transforms the IFC statements into RDF statements conforming to the PROPS, BOT and MEP ontologies.

2. BMS and IoT Metadata driver: Semantics are extracted from various metadata sources like mapping tables, vendor specifications, etc. to generate a metadata schema [91]. Here, ontologies such as Brick and SSN can be used to describe the BMS and IoT domains, respectively. Unique identifiers for devices or data streams can be used to link these metadata with their corresponding time series data, for instance using the `ref:hasTimeseriesId` relationship. Finally, these graphs are stored in a graph database and exposed via an API to interact with other services.

e: API SOFTWARE

The final component of the integration software is the API. After creating databases and core services, these are exposed to internal and external services via API software. The API provides a standardised way of accessing data and services towards its users, for instance, with REST or GraphQL over HTTP. In addition, security, access control, and monitoring are added at API level, avoiding the necessity to implement these at each service's individual level.

5) INFRASTRUCTURE SERVICES

These include a software platform's underlying containerisation, container orchestration, and infrastructure layers. These are not domain specific for smart buildings and rely on best practices. When considering NFR2 (reusability) and NFR2 (modularity), containers are inherently modular and enable packaging software components and services, making it easier to reuse them across different smart building applications. In terms of NFR3 (scalability), as the number of devices, data sources, or services in the building increases, container orchestration can efficiently handle the dynamic scaling of modules or services to accommodate the growing needs.

6) SHARED SERVICES

The functionality common across all services can be implemented as shared services. This includes security, governance, monitoring, and automation. Again, these are also not specific to smart buildings and rely on the industry best practices.

7) USER INTERFACES

Included here are user interfaces such as web or mobile Digital Twin applications, dashboards, charts, etc., that can

provide meaningful ways to interact with the data and the necessary access to the end user.

E. STEP 5 : SOLUTION ARCHITECTURE

The above software components constitute the solution architecture. In the literature review, we noted that there are multiple views for this architecture such as: business, technical and deployment. However, this study focuses primarily on the technical architecture. Fig. 6 shows the resulting technical architecture. This technology architecture can be found in the third row of the Zachman framework in Table 3. The seven software components identified above and their internal services are illustrated here, showing how the services of the seven components come together to make an overall system architecture.

F. STEP 6 : IMPLEMENTATION

This section presents the implementation of one possible instance of the proposed technical architecture and three use cases as example smart building applications. The use cases and the corresponding use case buildings (Living Labs) are as follows:

1) **Use Case 1: A Digital Twin application for the Atlas campus building at Eindhoven University of Technology.** Digital Twin applications involve integrating sensor data (both historical and real-time) with a digital model of a building [92], [93], [94], [95], which has rarely been achieved in the existing applications in a modular and reusable manner. This is often due to the complexity related to handling heterogeneous systems as described earlier. Most of the demonstrations [96], [97], [98] use Revit and Dynamo Visual Programming Language for historical sensor data integration. The downside of using commercial off-the-shelf software is that it is vendor-specific, so the reusability and extensibility are limited. In contrast, we propose a vendor-agnostic method for sensor data integration with BIM [22]. We use time series storage to record the historical sensor data. Metadata which describes BIM and BMS systems is kept in the graph database. SPARQL Protocol and RDF Query Language (SPARQL) queries executed against the metadata graph provide the necessary relationships (spaces, sensor IDs) to link the digital BIM model with sensor data. That enables a modular and reusable approach for creating Digital Twins.
2) **Use Case 2: Integration of real-time sensor data for the Atlas campus building at Eindhoven University of Technology.** Another essential component of a Digital Twin is real-time data. For demonstration purposes, real-time sensor data from multiple IoT sensors such as IEQ (temperature, humidity, light, TVOC and CO_2) and real-time energy consumption data (power, current) are used here. This application aims to integrate real-time IoT sensor data coming from heterogeneous sources, enrich them with semantics and visualise them in a web application.
3) **Use Case 3: Integration of a semantic building graph with BMS sensor data for Building 33 at Delft University of Technology.** One of the primary objectives of this use case building is to provide historical and real-time access to BMS sensor data for the development of data-driven smart building applications. However, the sensor data and metadata are not understood properly by users by default because of ambiguous semantics used by the vendor. To ensure the understanding of the sensor data, contextual data was first standardised using smart building ontologies. This contextual data enables sensor data filtering by equipment and sensor type. To improve the usability, we present the integration using Grafana dashboards, where semantic and time series data can be queried in combination.

The following section describes the implementation of various services that belong to the seven categories we proposed, namely, 1) existing business applications, 2) new microservice-based applications, 3) databases, 4) integration software, 5) infrastructure services, 6) shared services, and 7) user interfaces.

1) EXISTING BUSINESS APPLICATIONS

We rely on the BMS, BIM and IoT systems for the implementations. The current purpose of the interaction with the BMS is only to extract historical data. Existing BIM models, developed using Revit, are used to create Digital Twin applications. Other source systems include IoT devices which provide real-time sensor data. To integrate these existing systems, we use multiple drivers, which are described in the Integration Software section.

2) NEW MICROSERVICES-BASED APPLICATIONS

New applications reside primarily in the integration software and new business applications categories. At this stage, our services include IoT drivers, BMS drivers, file converters and ETL tools that provide data to the three use cases. Other planned services include data-driven applications such as MPC for DSM in buildings.

3) DATABASES

For the implementation, we rely on the four types of databases described in Section V. MongoDB time series collection is used to store time series data, whereas MongoDB document collection is used for storing other data regarding users and projects. GraphDB database stores the semantic graphs describing metadata of the buildings. As an object storage we use Minio to store IFC files, converted xkt files, and large time series data collections, etc. All databases are installed in the Docker environment.

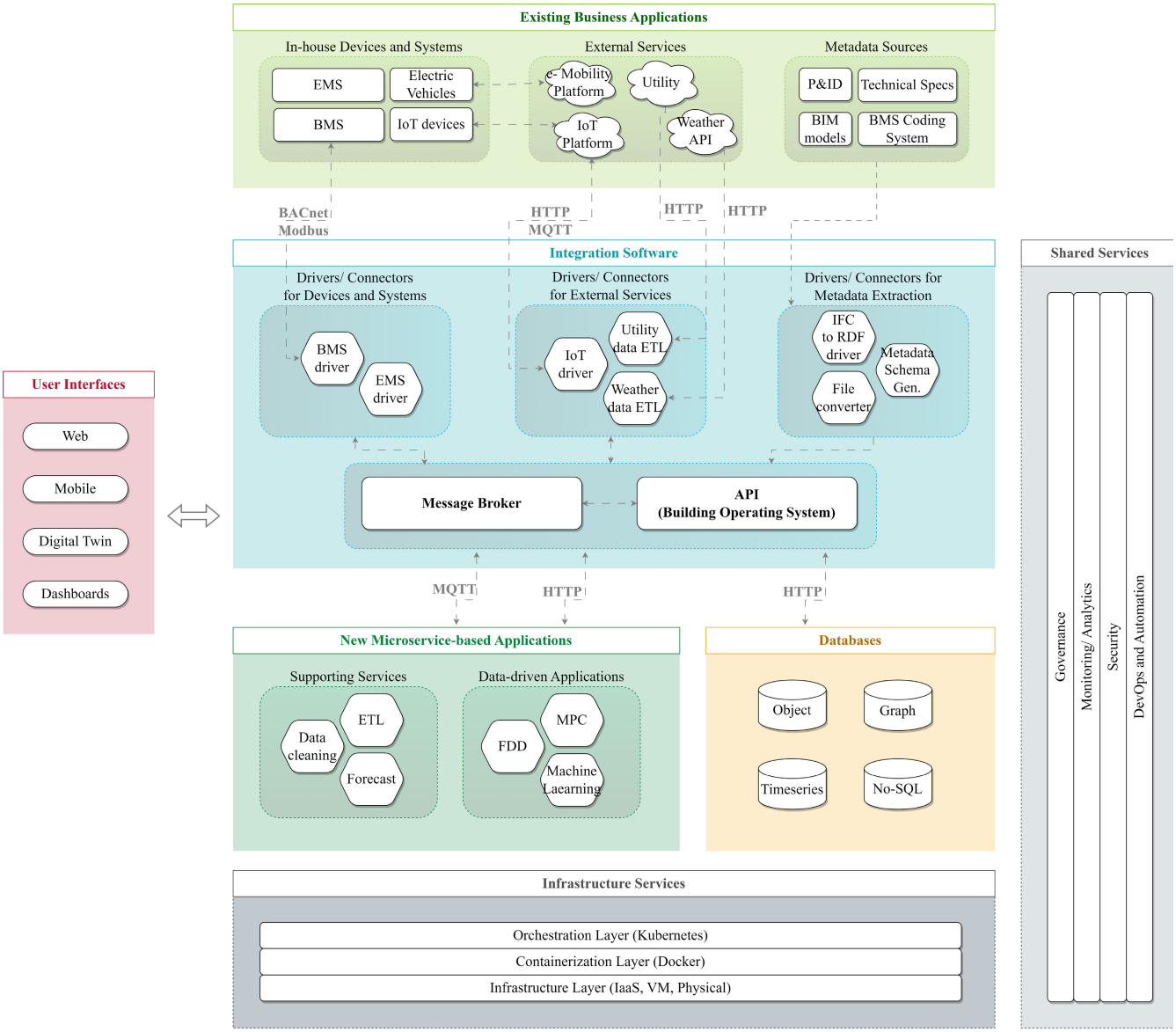

FIGURE 6. Proposed service-oriented software architecture for a data-driven smart building.

```
{
    "_id": ObjectId("6124567890abcdef12345678"),
    "sensor_id": "11NR00STE-001TRLL",
    "timestamp": ISODate("2023-07-28T12:00:00Z"),
    "value": 28.5
}
```

Listing 2. A time series data record stored in MongoDB time series collection.

```
{
    "_id": ObjectId("64dcfe2a7de01fc9ada366e6"),
    "title":"TU Eindhoven",
    "description":"Atlas Building",
    "xktFiles":["atlas.xkt"],
    "rdfFiles":[""atlas.ttl"],
    "queries":[],
    "__v":1
}
```

Listing 3. A project data record stored in MongoDB collection.

An example of the physical data model of a MongoDB document stored in the time series data collection for Use Case 1 (Digital Twin) is shown in Listing 2:

The physical data model of the project data stored in MongoDB collection for Use Case 1 (Digital Twin) is shown in Listing 3:

A part of the semantic building graph in RDF syntax stored in GraphDB database for Use Case 1 and 2 (Digital Twin and IoT integration) is shown in Listing 4. This graph utilises the Brick ontology version v1.3. The same graph is visually represented in Fig. 7.

4) INTEGRATION SOFTWARE

a: BMS INTEGRATION

At this stage, we use historical data from the BMS for Use Case 1 and Use Case 3. An ETL process is used to transform the incoming data into the MongoDB schema defined in the

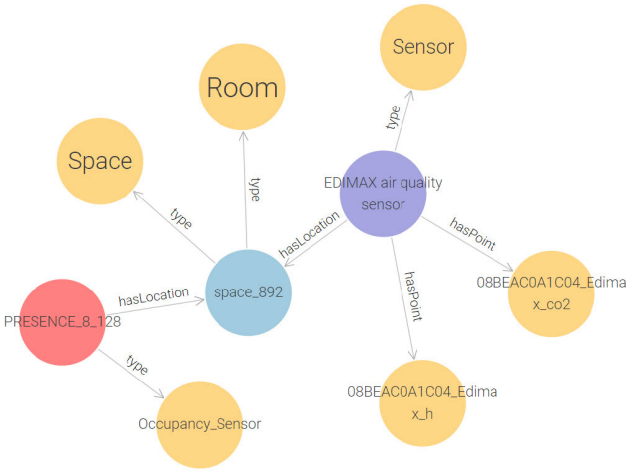

FIGURE 7. Visual representation of the semantic graph using GraphDB interface. The graph shows the relationships between spaces and sensors to enable the Digital Twin and IoT Integration.

```
inst:08BEAC0A1C04_Edimax a brick:Sensor ;
rdfs:label "EDIMAX air quality sensor";
  brick:hasPoint
    inst:08BEAC0A1C04_Edimax_co2,
    inst:08BEAC0A1C04_Edimax_h ;
    brick:hasLocation inst:space_892 ;
ref:hasExternalReference [
        a ref:TimeseriesReference ;
        ref:hasTimeseriesId "edimax/08BEAC0A1C04/pub" ;
        ref:storedAt inst:mongo-timeseries ;
    ] .

inst:11NR00STE-001TRL_Sensor a brick:Temperature_Sensor ;
    rdfs:label "TEMPERATURE_8_128" ;
    brick:hasLocation inst:space_892 ;
ref:hasExternalReference [
        a ref:TimeseriesReference ;
        ref:hasTimeseriesId "11NR00STE-001TRL" ;
        ref:storedAt inst:mongo-timeseries ;
    ] .

inst:space_892
        a bot:Space, brick:Room ;
        bot:hasGuid "1456e859-2e9e-4e70-9341-f073adb2dbb0";
        props:hasCompressedGuid "0KLkXPBfvES9D1y7Ejijkm" .

 inst:mongo-timeseries a ref:Database ;
    rdfs:label "MongoDB Timeseries Collection" .
```

Listing 4. A part of the semantic graph stored in RDF format in GraphDB graph database.

logical data model. Protocol-specific drivers are needed for future developments to establish the bidirectional communication required for applications like MPC.

b: IoT INTEGRATION

To demonstrate the IoT integration, we used three types of IoT devices. The first device is an air quality sensor, the data from which is accessed in real-time by querying the platform's API. The IoT driver for this device fetches sensor data every 10 seconds from the IoT platform and publishes them to the MQTT broker. The second device is a smart socket. It uses a proprietary protocol, and therefore, the IoT driver for this device adapts an already available connector [99] to forward these messages to an MQTT broker. The third IoT device is custom-made by the authors and directly communicates to the MQTT broker. This way, all real-time data becomes accessible via the single MQTT message broker. Each IoT device ID (optionally MAC address, if available) is included in the MQTT topic, and the sensor readings are included in the MQTT payload. The payload of each IoT device is processed using the service previously demonstrated in Fig. 5. The physical data model of the air quality sensor resulting from the service is shown in Listing 5. This data model is compatible with many IoT payloads. This unified data is also recorded in a time series database for later usage.

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

Listing 5. Logical data model of the processes messages.

c: BIM INTEGRATION

BIM integration is used in Use Case 1 in the Atlas building. For BIM integration, we rely on the xeokit SDK. This implementation builds on previous work describing the implementation of a web application to visualise a web-based BIM model using xeokit [22]. In this case, another driver is required to convert IFC files to the web-compatible xkt format. This conversion is achieved using the conversion process elaborated in ifc2xkt [89]. The converted files are stored in an object storage which can be accessed via the API.

d: METADATA INTEGRATION

All use cases described above rely on the metadata integration service. The goal of the metadata graph is to standardise the semantics and create a link between different data sources. The three most critical data sources are the BMS, IoT and the BIM model, as they contain a large amount of metadata that enables contextual representations in data-driven applications. In this regard, two types of metadata drivers were implemented, one for the IFC-to-RDF conversion and another for generating a metadata schema based on BMS and IoT sensors.

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. This study achieves this through the implementation of an IFC to RDF driver. In the IFC-to-RDF conversion, the semantic model of the building is generated based on the IFC file using an IFC-to-RDF converter [90]. It transforms the IFC statements into RDF statements conforming to the PROPS, BOT and MEP ontologies.

Brick and SSN ontologies are used to describe the BMS and IoT domains respectively. Semantics are extracted from the metadata sources like data dictionaries and device specifications in BMS to create a metadata schema [91]. These graphs are stored in the GraphDB graph database. Discovering the sensors and observations using their sensor type, location, or other relationships is realised by executing

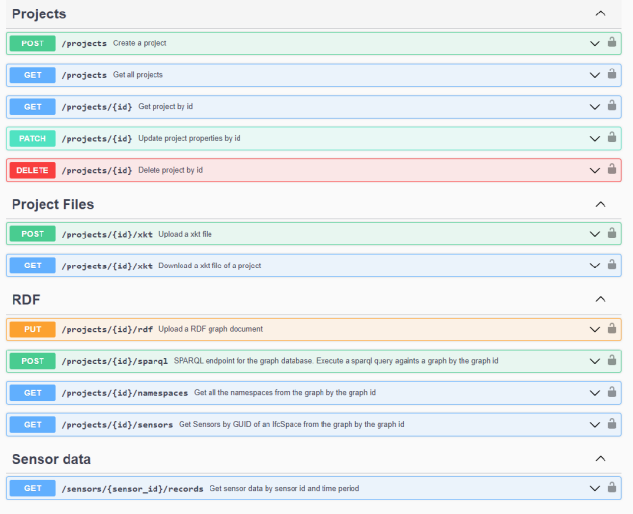

FIGURE 8. Sample API endpoints shown in openAPI specification, for interacting with the various services, sensor data and metadata schema of the building.

SPARQL queries on the graph database via the API. Reusable SPARQL queries are programmed into the API to improve the overall reusability of the application. A part of the metadata schema with building and sensor semantics of the Atlas Living Lab is shown in Listing. 4, and a graphical version is shown in Fig. 7.

e: API

The purpose of the REST API is to facilitate the communication between the smart building applications and the other components in the architecture. All three use cases use the same API. The objective is to keep this API more general and applicable to any data-driven building, not specific to one use case building.

The API is implemented in Node.js using Nest.js framework, and sample endpoints in openAPI specification are shown in Fig. 8. It allows applications to interact with various services and execute Create-Read-Update-Delete (CRUD) operations against the available four databases. Time series data can be accessed via the API by querying with the relevant unique sensor ID. The API also provides endpoints to query the graphs in the RDF database, the project data stored in the document database, and the files stored in the Minio storage. Future developments consider other data-driven services such as forecasting deployed as a service.

The following section discusses an example interaction with the API.

The first scenario is querying a sensor's time series data using its metadata. The utilised metadata is a known space identifier (e.g., IFC GUID of a space). In this case, we need to execute a SPARQL query as shown in Listing 6, which enables getting the sensors in a particular space. It returns the identifiers of the sensors contained in a given space using *brick:hasPoint* relationship. These Points contain the time series identifiers used to record the corresponding sensor data in the time series database.

```
SELECT ?space ?location ?sensor_id ?sensor_type ?tsid
WHERE {
 VALUES ?value {
      ${spaces.map(space => ''${space}'').join(' ')}
}
?location props:hasCompressedGuid ?value .
?location props:hasCompressedGuid ?space .
?item brick:hasLocation ?location .
?item brick:hasPoint ?sensor_id .
?item a ?sensor_type .
    OPTIONAL{
        ?item ref:hasExternalReference ?arr .
        ?arr ref:hasTimeseriesId ?tsid .
    }
}
```

Listing 6. SPARQL query to get all sensors in a space.

```
Method          : POST
URL             : /rdf/{id}/sensors
Request body    : {
  "spaces": [
    "0KLkXPBfvES9D1y7Ejijkm"
  ]}
Response        :
 [
  {
    "space": {
      "type": "literal",
      "value": "0KLkXPBfvES9D1y7Ejijkm"
    },
    "location": {
      "type": "uri",
      "value": "inst:space_892"
    },
    "sensor_id": {
      "type": "uri",
      "value": "inst:11NR00STE-001TRL_Sensor"
    },
    "sensor_type": {
      "type": "uri",
      "value": "#Brick#Temperature_Sensor"
    },
    "tsid": {
      "type": "literal",
      "value": "11NR00STE-001TRL"
    }
  },
  .
  .
 ]
```

Listing 7. Get sensors by GUID of space.

Then, these time series identifiers can be used to retrieve relevant sensor data from the time series storage as shown in Listing 8.

The execution of these kind of queries are encapsulated in the API endpoints to improve reusability. For example, Listing 7 shows an API request that takes the GUID of the space and returns the sensors in it.

5) INFRASTRUCTURE SERVICES

We use Docker as the containerisation environment, as newer applications, such as IoT drivers, APIs, and databases, run on Docker.

6) SHARED SERVICES

The functionalities common across all services are recognised as shared services. These include security, governance, monitoring, and automation. Again, these are also not specific to smart buildings.

```
Method           : GET
URL              : /sensors/{id}/records
Query parameters: sensor_id, start date, end date
Response         : {
  "sensor_id": "11NR00STE-001TRL",
  "start": "2021-01-01T10:06:03.837Z",
  "end": "2022-01-01T10:06:03.837Z",
  "result": [
    {
      "timestamp": "2021-04-30T20:02:00.000Z",
      "value": 22
    },
    {
      "timestamp": "2021-04-30T11:02:00.000Z",
      "value": 21.9
    },
    {
      "timestamp": "2021-04-30T08:02:00.000Z",
      "value": 22.4
    },
    .
    .
    ]
}
```

Listing 8. Get sensor records by sensor id.

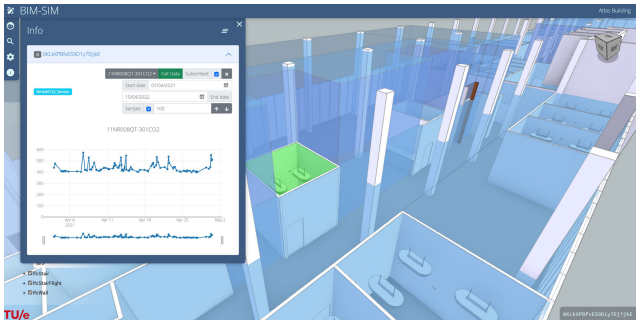

FIGURE 9. Integration of historical CO_2 sensor data of the selected space during April 2021 and May 2021 [22].

7) USER INTERFACES

This section demonstrates the implementations of two web applications for Use Case 1 and 2, and a Grafana dashboard for Use Case 3. All applications are developed based on the proposed and implemented architecture.

The first user interface is developed for Use Case 1 in the Atlas building using the React front-end framework and the xeokit SDK. The application allows the user to navigate through the BIM model using the web browser and see the sensor data related to its spaces. The REST API connects the web application to the semantic graph, time series data (both historical and real-time) and BIM models. Fig. 9 demonstrates historical sensor data linked to a BIM model and visualised in the web application [22]. Fig. 9 also shows how the historical data is retrieved for a CO_2 sensor installed inside a room on the 8th floor of the Atlas building.

Fig. 10 shows the second application developed for Use Case 2. This application was also developed using the React front-end framework. The application allows the user to interact with the metadata graph of the Atlas building, explore its IoT devices and receive their real-time data [87]. This application relies on the IoT drivers we implemented to unify the IoT payloads from three heterogeneous sources.

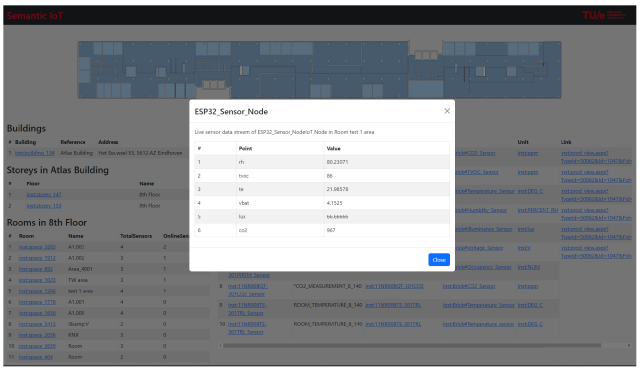

FIGURE 10. Integration of real-time sensor data using metadata graph [87].

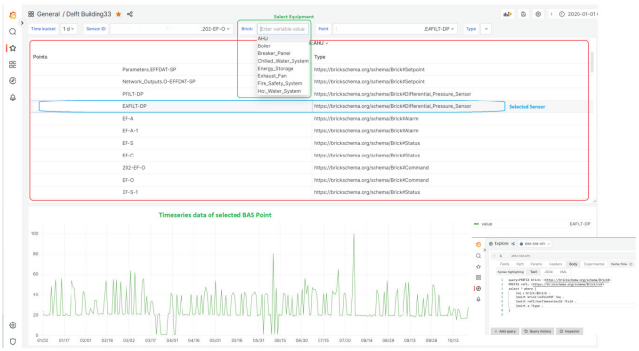

FIGURE 11. BMS time series data exploration using metadata schema and Grafana dashboards [91].

The third application is developed as a data exploratory and visualisation tool for BMS data in Use Case 3. We developed a metadata schema for the BMS and integrated it with the time series database via a Grafana dashboard using the API, providing the ability to query time series data efficiently. That includes the ability to query data by equipment or point types and gain initial insights about the data through charts. Users can query the BMS points of interest by selecting the equipment type from the drop-down list in Fig. 11. This list contains the equipment types from the BMS, annotated according to the Brick ontology.

Fig. 11 shows an example of an Air Handling Unit (AHU) selected as *brick:Equipment*. Then, a SPARQL query requests the unique identifiers of the data points related to this *brick:AHU* in the metadata schema. The user can select one or more points from the resulting point list to explore their time series data. Overall, the architecture enables efficient means to understand and explore the large amounts of time series data available from the building.

V. CONCLUSION AND FUTURE WORK

A. CONCLUSION

Ambitious operational objectives related to building energy efficiency, energy flexibility, IEQ, occupant comfort and well-being, etc., call for efficient data-driven smart building applications. These applications demand seamless data

linking and integration, and their underlying systems must also be integrated to exchange information. However, current system architectures in the data-driven building domain are inadequate for smart building applications because they 1) lack the integration of best practices such as architecture frameworks and mature architecture patterns, 2) do not rely on semantic technologies as an architectural component and have showcased limited capabilities using ad-hoc tools, and 3) are largely conceptual and practical implementations are rare. On the other hand, well-tested architecture patterns available in other IT domains cannot be directly transferred to a data-driven building context due to the lack of domain-specific components.

In response to the above shortcomings, this study proposes a service-oriented architecture for the data-driven building domain. The proposed architecture relies on the Zachman architecture framework, functional and non-functional requirements derived by interviewing industry experts and literature, and already available, well-tested architecture patterns in the IT industry. The study demonstrates notable improvements in scalability, modularity and reusability due to the utilised MACH architecture principles and ontology-based extensible metadata schemes. The implementation includes specific architectural components for the data-driven building domain. We also implement multiple drivers/connectors in response to the diverse information sources that need to be integrated with new microservices-based applications. Those drivers include BMS, IoT and metadata conversion drivers. The new smart building applications follow a microservice architecture. The proposed API and message bus establish the communication between microservices and the other components. The architecture leverages semantic technologies as an integral part of the proposed API by programming reusable SPARQL queries into API endpoints. Furthermore, we show how different data types should be maintained in different databases (e.g., time series and graph databases) for efficient querying. The original data types are retained in their optimum environments by choosing appropriate databases. In contrast to custom data models which are difficult to extend or reuse, we utilise open metadata models and standards, as well as well-established ontologies such as Brick, SSN, BOT, etc. This way, the data models remain flexible and extensible.

Furthermore, the article demonstrates the implementation of an instance of the architecture and three smart building applications, validating the reusability and modularity of the proposed architecture. For example, the presented Digital Twin application for the Atlas use case building can be easily extended and used in another building. Having created a metadata schema and BIM model of the new building, it can use the same application logic (API) and the services, and, therefore, no significant additional effort is required to configure the application. That also applies to the other two applications demonstrated in the implementation section. Other applications, such as MPC, are currently under implementation as new microservices.

Implementing an instance of the proposed architecture and the three smart building applications reveals the architecture's suitability as a blueprint to create similar system architectures for smart buildings. Besides research, this article is also relevant for solution architects who want to understand the building domain, and business executives and owners looking to build solution architectures for smart buildings. This type of architecture is also useful for service providers who develop and provide software for smart buildings. For other service providers, such as energy companies or algorithm developers, who wish to interact with the smart building, this software architecture provides a standard API helping to make the data and systems accessible and avoiding any ambiguities when implementing new smart building applications. A building is a complex ecosystem with many different vendors, protocols, data models, naming conventions, etc. Therefore, the practical end-to-end implementation of this software architecture for a fully functional smart building is non-trivial and involves several stakeholders, software architects, developers and deployment infrastructure.

B. FUTURE WORK

Implementing DSM applications such as MPC as microservice-based applications using the proposed architecture is currently under implementation. To that end, additional services for data analytics, Machine Learning, forecasting, and exploratory data analysis tools can also be implemented as new services. The proposed service-oriented architecture allows creating services required for MPC as modular and reusable components. Sophisticated control strategies such as MPC usually require combining diverse data sources. The proposed architecture is well-suited for such integration. These applications can be developed and deployed individually in containerised environments, relying on the core of the system architecture, the API and the message broker for communication.

Utilising semantic technologies as an integral part of the architecture is essential for adding context to operational data and creating links between diverse systems. However, generating a metadata schema from available metadata sources can be quite a laborious process, especially if the metadata is not well-documented or not available in structured formats. The efficient generation of metadata graphs itself is an emerging research area, which tries to automate the process [14], [100], and improve the semantic sufficiency of the generated graph for a particular application [101]. When mature, these services should be integrated with the proposed architecture.

Although the three implemented smart building applications do not rely on a bi-directional communication with the building controllers, other applications, such as FDD and MPC, rely on such active bi-directional communication. The idea is to control the devices and controllers using intelligent algorithms that send optimal set points and control commands to the device APIs via relevant drivers. Using an event-driven architecture [102] that uses events to share information between decoupled services is most suitable for these types of

coordinated control applications. The current implementation of the architecture relies on a REST API and a message broker for sharing information between systems. An event-driven service component can also be added to the architecture in the future. This addition is valuable for continuously monitoring the devices or services and triggering the relevant services that rely on those events.

It is also essential to investigate how data can be categorised into internal, shared and open data since the privacy and security aspects can differ. A standardised methodology for secure and ethical access, use of data (collection, management and use) and control of devices, is also under investigation [19].

REFERENCES

[1] P. Arjunan, M. Srivastava, A. Singh, and P. Singh, "OpenBAN: An open building analytics middleware for smart buildings," *EAI Endorsed Trans. Scalable Inf. Syst.*, vol. 2, no. 7, p. e4, Aug. 2015.

[2] B. Balaji, A. Bhattacharya, G. Fierro, J. Gao, J. Gluck, D. Hong, A. Johansen, J. Koh, J. Ploennigs, Y. Agarwal, M. Bergés, D. Culler, R. K. Gupta, M. Srivastava, and K. Whitehouse, "Brick: Metadata schema for portable smart building applications," *Appl. Energy*, vol. 226, pp. 1273–1292, Sep. 2018.

[3] X. Gao, P. Pishdad-Bozorgi, D. R. Shelden, and S. Tang, "Internet of Things enabled data acquisition framework for smart building applications," *J. Construct. Eng. Manage.*, vol. 147, no. 2, Feb. 2021, Art. no. 04020169.

[4] R. Cox, S. Walker, J. van der Velden, P. Nguyen, and W. Zeiler, "Flattening the electricity demand profile of office buildings for future-proof smart grids," *Energies*, vol. 13, no. 9, p. 2357, May 2020.

[5] A. Krishnan Prakash, K. Zhang, P. Gupta, D. Blum, M. Marshall, G. Fierro, P. Alstone, J. Zoellick, R. Brown, and M. Pritoni, "Solar+ optimizer: A model predictive control optimization platform for grid responsive building microgrids," *Energies*, vol. 13, no. 12, p. 3093, Jun. 2020.

[6] S. Lazarova-Molnar, H. R. Shaker, and N. Mohamed, "Fault detection and diagnosis for smart buildings: State of the art, trends and challenges," in *Proc. 3rd MEC Int. Conf. Big Data Smart City (ICBDSC)*, Mar. 2016, pp. 1–7.

[7] M. S. Mirnaghi and F. Haghighat, "Fault detection and diagnosis of large-scale HVAC systems in buildings using data-driven methods: A comprehensive review," *Energy Buildings*, vol. 229, Dec. 2020, Art. no. 110492.

[8] M. Bird, C. Daveau, E. O'Dwyer, S. Acha, and N. Shah, "Real-world implementation and cost of a cloud-based MPC retrofit for HVAC control systems in commercial buildings," *Energy Buildings*, vol. 270, Sep. 2022, Art. no. 112269.

[9] D. Blum, Z. Wang, C. Weyandt, D. Kim, M. Wetter, T. Hong, and M. A. Piette, "Field demonstration and implementation analysis of model predictive control in an office HVAC system," *Appl. Energy*, vol. 318, Jul. 2022, Art. no. 119104.

[10] Z. Chevallier, B. Finance, and B. C. Boulakia, "A reference architecture for smart building digital twin," in *Proc. SeDiT@ ESWC*, 2020, pp. 1–12.

[11] G. B. Ozturk, "Digital twin research in the AECO-FM industry," *J. Building Eng.*, vol. 40, Aug. 2021, Art. no. 102730.

[12] S. Aheleroff, X. Xu, R. Y. Zhong, and Y. Lu, "Digital twin as a service (DTaaS) in industry 4.0: An architecture reference model," *Adv. Eng. Informat.*, vol. 47, Jan. 2021, Art. no. 101225.

[13] P. Pauwels, S. Zhang, and Y.-C. Lee, "Semantic web technologies in AEC industry: A literature overview," *Autom. Construct.*, vol. 73, pp. 145–165, Jan. 2017.

[14] A. A. Bhattacharya, D. Hong, D. Culler, J. Ortiz, K. Whitehouse, and E. Wu, "Automated metadata construction to support portable building applications," in *Proc. 2nd ACM Int. Conf. Embedded Syst. Energy-Efficient Built Environ.* New York, NY, USA: ACM, Nov. 2015, pp. 3–12.

[15] Y. Agarwal, R. Gupta, D. Komaki, and T. Weng, "BuildingDepot," in *Proc. 4th ACM Workshop Embedded Sens. Syst. Energy-Efficiency Buildings*, New York, NY, USA, Nov. 2012, pp. 64–71.

[16] 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, p. 2568, Apr. 2022.

[17] C. Fernando, "Industry-specific architecture patterns," in *Solution Architecture Patterns for Enterprise: A Guide to Building Enterprise Software Systems*. Berkeley, CA, USA: Apress, 2023, ch. 9, pp. 313–359, doi: 10.1007/978-1-4842-8948-8_9.

[18] J. A. Zachman, "A framework for information systems architecture," *IBM Syst. J.*, vol. 38, no. 2, pp. 454–470, 1999.

[19] *Work Packages—Brains4buildings*. Accessed: Aug. 7, 2023. [Online]. Available: https://brains4buildings.org/work-packages/

[20] M. Richards, *Software Architecture Patterns*, 2nd ed. Sebastopol, CA, USA: O'Reilly Media, Inc., 2022. [Online]. Available: https://www.oreilly.com/library/view/software-architecture-patterns/9781491971437

[21] *Enterprise MACHified | MACH Alliance*. Accessed: Aug. 7, 2023. [Online]. Available: https://machalliance.org/

[22] L. Chamari, E. Petrova, and P. Pauwels, "A web-based approach to BMS, BIM and IoT integration: A case study," in *Proc. REHVA 14th HVAC World Congr.*, 2022, pp. 1–8.

[23] N. Medvidovic and R. N. Taylor, "Software architecture: Foundations, theory, and practice," in *Proc. ACM/IEEE 32nd Int. Conf. Softw. Eng.*, vol. 2. New York, NY, USA, May 2010, pp. 471–472.

[24] M. Guessi, V. V. G. Neto, T. Bianchi, K. R. Felizardo, F. Oquendo, and E. Y. Nakagawa, "A systematic literature review on the description of software architectures for systems of systems," in *Proc. 30th Annu. ACM Symp. Appl. Comput.*, New York, NY, USA, Apr. 2015, pp. 1433–1440.

[25] C. Fernando, "Introduction to solution architecture," in *Solution Architecture Patterns for Enterprise: A Guide to Building Enterprise Software Systems*. Berkeley, CA, USA: Apress, 2023, ch. 2, pp. 29–61, doi: 10.1007/978-1-4842-8948-8_2.

[26] S. Bhattarai, G. Bleakley, M. Buchheit, C. Byers, A. Chigan, M. Crawford, J. Durand, and A. Karmarkar. (2022). *The Industrial Internet Reference Architecture*. Industrial IoT Consortium. [Online]. Available: https://www.iiconsortium.org/IIRA/

[27] The Open Group Standard. (2022). *The TOGAF Standard*. Accessed: Aug. 28, 2023. [Online]. Available: http://www.opengroup.org/togaf/

[28] K. Smolander, "What is included in software architecture? A case study in three software organizations," in *Proc. 9th Annu. IEEE Int. Conf. Workshop Eng. Comput.-Based Syst.*, 2002, pp. 131–138.

[29] J. A. Zachman, "The Zachman framework for enterprise architecture, primer for enterprise engineering and manufacturing," *Zachman Int.*, vol. 128, no. 9, p. 15, 2003. [Online]. Available: https://www.dragon1.com/downloads/ZachmanBookRFIextract.pdf

[30] T. Ylimäki and V. Halttunen, "Method engineering in practice: A case of applying the Zachman framework in the context of small enterprise architecture oriented projects," *Inf., Knowl., Syst. Manage.*, vol. 5, no. 3, pp. 189–209, 2005.

[31] A. Radwan and M. Aarabi, "Study of implementing Zachman framework for modeling information systems for manufacturing enterprises aggregate planning," in *Proc. Int. Conf. Ind. Eng. Oper. Manag.*, 2011, pp. 9–14.

[32] H. Tannady, J. F. Andry, B. G. Sudarsono, and Y. Krishartanto, "Enterprise architecture using Zachman framework at paint manufacturing company," *Technol. Rep. Kansai Univ.*, vol. 62, no. 4, pp. 1869–1883, 2020.

[33] J. M. Nogueira, D. Romero, J. Espadas, and A. Molina, "Leveraging the Zachman framework implementation using action—Research methodology—A case study: Aligning the enterprise architecture and the business goals," *Enterprise Inf. Syst.*, vol. 7, no. 1, pp. 100–132, Feb. 2013.

[34] S. Bondar, J. C. Hsu, A. Pfouga, and J. Stjepandić, "Agile digital transformation of system-of-systems architecture models using Zachman framework," *J. Ind. Inf. Integr.*, vol. 7, pp. 33–43, Sep. 2017.

[35] R. Alm and M. Wißotzki, "TOGAF adaption for small and medium enterprises," *Lect. Notes Bus. Inf. Process.*, vol. 160, pp. 112–123, 2013.

[36] S. Kotusev, "TOGAF-based enterprise architecture practice: An exploratory case study," *Commun. Assoc. Inf. Syst.*, vol. 43, no. 1, pp. 321–359, 2018.

[37] B. Anthony, S. A. Petersen, D. Ahlers, J. Krogstie, and K. Livik, "Big data-oriented energy prosumption service in smart community districts: A multi-case study perspective," *Energy Informat.*, vol. 2, no. 1, p. 36, Dec. 2019.

[38] W. Engelsman and R. Wieringa, "Goal-oriented requirements engineering and enterprise architecture: Two case studies and some lessons learned," in *Proc. Int. Work. Conf. Requirement Eng., Found. Softw. Quality*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 7195, 2012, pp. 306–320.

[39] OMG. (2017). *About the Unified Modeling Language Specification Version 2.5.1*. Accessed: Jul. 3, 2023. [Online]. Available: https://www.omg.org/spec/UML

[40] The Open Group Standard. (2017). *ArchiMate® 3.2 Specification*. [Online]. Available: https://pubs.opengroup.org/architecture/archimate3-doc/

[41] M. Lankhorst, "Viewpoints and visualisation," in *Enterprise Architecture at Work*. Berlin, Germany: Springer, 2012, ch. 8, pp. 147–188, doi: 10.1007/978-3-662-53933-0_8.

[42] I. Malavolta, P. Lago, H. Muccini, P. Pelliccione, and A. Tang, "What industry needs from architectural languages: A survey," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 869–891, Jun. 2013.

[43] P. B. Kruchten, "The 4+1 view model of architecture," *IEEE Softw.*, vol. 12, no. 6, pp. 42–50, Mar. 1995.

[44] Atlassian. *Microservices vs. Monolithic Architecture*. Accessed: Aug. 7, 2023. [Online]. Available: https://www.atlassian.com/microservices/microservices-architecture/microservices-vs-monolith

[45] A. Kharenko. *Monolithic vs. Microservices Architecture*. Accessed: Aug. 7, 2023. [Online]. Available: https://articles.microservices.com/monolithic-vs-microservices-architecture-5c4848858f59

[46] AWS. *What's the Difference Between Monolithic and Microservices Architecture?* Accessed: Aug. 7, 2023. [Online]. Available: https://aws.amazon.com/compare/the-difference-between-monolithic-and-microservices-architecture/

[47] D. Shadija, M. Rezai, and R. Hill, "Towards an understanding of microservices," in *Proc. 23rd Int. Conf. Autom. Comput. (ICAC)*, Sep. 2017, pp. 1–6.

[48] H. Calderón-Gómez, L. Mendoza-Pittí, M. Vargas-Lombardo, J. M. Gómez-Pulido, D. Rodríguez-Puyol, G. Sención, and M.-L. Polo-Luque, "Evaluating service-oriented and microservice architecture patterns to deploy eHealth applications in cloud computing environment," *Appl. Sci.*, vol. 11, no. 10, p. 4350, May 2021.

[49] M. Richards, *Microservices vs. Service-Oriented Architecture*. Berlin, Heidelberg: O'Reilly Media, 2016. [Online]. Available: https://www.oreilly.com/library/view/microservices-vs-service-oriented/9781491975657/

[50] M. P. Papazoglou and W.-J. van den Heuvel, "Service oriented architectures: Approaches, technologies and research issues," *VLDB J.*, vol. 16, no. 3, pp. 389–415, Jul. 2007.

[51] R. Perrey and M. Lycett, "Service-oriented architecture," in *Proc. Symp. Appl. Internet Workshops*, 2003, pp. 116–119.

[52] I. Jerstad, S. Dustdar, and D. V. Thanh, "A service oriented architecture framework for collaborative services," in *Proc. 14th IEEE Int. Workshops Enabling Technol., Infrastruct. Collaborative Enterprise*, Jun. 2005, pp. 121–125.

[53] M. Rychlý and P. Weiss, "Modeling of service oriented architecture—From business process to service realisation," in *Proc. 3rd Int. Conf. Eval. Novel Approaches Softw. Eng.*, 2008, pp. 140–146.

[54] M. Luo, B. Goldshlager, and L.-J. Zhang, "Designing and implementing enterprise service bus (ESB) and SOA solutions," in *Proc. IEEE Int. Conf. Services Comput. (SCC)*, vol. 2, Jul. 2005, pp. 1–2.

[55] C. Fernando, "Building enterprise software systems with microservice architecture," in *Solution Architecture Patterns for Enterprise: A Guide to Building Enterprise Software Systems*. Berkeley, CA, USA: Apress, 2023, ch. 3, pp. 63–108, doi: 10.1007/978-1-4842-8948-8_3.

[56] S. Katipamula and M. Brambley, "Review article: Methods for fault detection, diagnostics, and prognostics for building systems—A review, Part I," *HVAC R Res.*, vol. 11, no. 1, pp. 3–25, Jan. 2005.

[57] P. Delgoshaei and M. A. Austin, "Framework for knowledge-based fault detection and diagnostics (don't short) in multi-domain systems: Application to heating ventilation and air conditioning systems," *Int. J. Adv. Intell. Syst.*, vol. 10, no. 3, pp. 393–409, 2017.

[58] K. Zhang, A. Prakash, L. Paul, D. Blum, P. Alstone, J. Zoellick, R. Brown, and M. Pritoni, "Model predictive control for demand flexibility: Real-world operation of a commercial building with photovoltaic and battery systems," *Adv. Appl. Energy*, vol. 7, Sep. 2022, Art. no. 100099.

[59] K. Bao, I. Mauser, S. Kochanneck, H. Xu, and H. Schmeck, "A microservice architecture for the intranet of things and energy in smart buildings," in *Proc. 1st Int. Workshop Mashups Things APIs*, New York, NY, USA, Dec. 2016, pp. 1–6.

[60] S. Dawson-Haggerty, A. Krioukov, J. Taneja, S. Karandikar, G. Fierro, N. Kitaev, and D. Culler, "BOSS: Building operating system services," in *Proc. 10th USENIX Symp. Networked Syst. Design Implement.*, 2013, pp. 443–457.

[61] M. Jahn, D. Berichter, and M. Jarke, "Turning smart buildings into innovation environments," Ph.D. dissertation, Dept. Math., Comput. Sci. Natural Sci., RWTH Aachen Univ., Aachen, Germany, 2016.

[62] E. Patti, A. Acquaviva, M. Jahn, F. Pramudianto, R. Tomasi, D. Rabourdin, J. Virgone, and E. Macii, "Event-driven user-centric middleware for energy-efficient buildings and public spaces," *IEEE Syst. J.*, vol. 10, no. 3, pp. 1137–1146, Sep. 2016.

[63] T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquisition*, vol. 5, no. 2, pp. 199–220, Jun. 1993.

[64] M. H. Rasmussen, M. Lefrançois, G. F. Schneider, and P. Pauwels, "BOT: The building topology ontology of the W3C linked building data group," *Semantic Web*, vol. 12, no. 1, pp. 143–161, Nov. 2020.

[65] *Building Topology Ontology*. Accessed: Aug. 7, 2023. [Online]. Available: https://github.com/w3c-lbd-cg/bot

[66] *Semantic Sensor Network Ontology*. Accessed: Aug. 7, 2023. [Online]. Available: https://www.w3.org/TR/vocab-ssn/

[67] K. Janowicz, A. Haller, S. J. D. Cox, D. Le Phuoc, and M. Lefrançois, "SOSA: A lightweight ontology for sensors, observations, samples, and actuators," *J. Web Semantics*, vol. 56, pp. 1–10, May 2019.

[68] W. Terkaj, G. F. Schneider, and P. Pauwels, "Reusing domain ontologies in linked building data: The case of building automation and control," in *Proc. Joint Ontol. Workshops, Tyrolean Autumn Ontol.*, vol. 2050, 2017. [Online]. Available: https://ceur-ws.org/Vol-2050/

[69] *Home—BrickSchema*. Accessed: Aug. 9, 2023. [Online]. Available: https://brickschema.org/

[70] *Home—Project Haystack*. Accessed: Aug. 7, 2023. [Online]. Available: https://project-haystack.org/

[71] P. Delgoshaei, M. Heidarinejad, and M. A. Austin, "A semantic approach for building system operations: Knowledge representation and reasoning," *Sustainability*, vol. 14, no. 10, p. 5810, May 2022.

[72] P. Delgoshaei, M. A. Austin, and D. Veronica, "Semantic models and rule-based reasoning for fault detection and diagnostics: Applications in heating, ventilating and air conditioning systems," in *Proc. 12th Int. Conf. Syst.*, 2017, pp. 48–53.

[73] Y.-Y. Zhang, Z.-Z. Hu, J.-R. Lin, and J.-P. Zhang, "Linking data model and formula to automate KPI calculation for building performance benchmarking," *Energy Rep.*, vol. 7, pp. 1326–1337, Nov. 2021.

[74] H. Dibowski, J. Vass, O. Holub, and J. Rojicek, "Automatic setup of fault detection algorithms in building and home automation," in *Proc. IEEE 21st Int. Conf. Emerg. Technol. Factory Autom. (ETFA)*, Sep. 2016, pp. 1–6.

[75] O. Topcu and H. Oğuztüzün, "High level architecture," in *Guide to Distributed Simulation With HLA* (Simulation Foundations, Methods and Applications). Cham, Switzerland: Springer, 2017, ch. 2, pp. 29–78, doi: 10.1007/978-3-319-61267-6_2.

[76] E. Curry, A. Metzger, A. J. Berre, A. Monzón, and A. Boggio-Marzet, "A reference model for big data technologies," in *The Elements of Big Data Value*. Cham, Switzerland: Springer, 2021, pp. 127–151.

[77] M. Mazzara, I. Afanasyev, S. R. Sarangi, S. Distefano, V. Kumar, and M. Ahmad, "A reference architecture for smart and software-defined buildings," in *Proc. IEEE Int. Conf. Smart Comput. (SMARTCOMP)*, Jun. 2019, pp. 167–172.

[78] M. Abu-Matar, "Towards a software defined reference architecture for smart city ecosystems," in *Proc. IEEE Int. Smart Cities Conf.*, Sep. 2016, pp. 1–6.

[79] 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, Nov. 2022, Art. no. 092036.

[80] M. R. Bashir, A. Q. Gill, and G. Beydoun, "A reference architecture for IoT-enabled smart buildings," *Social Netw. Comput. Sci.*, vol. 3, no. 6, p. 493, Sep. 2022.

[81] M. R. Bashir, A. Q. Gill, G. Beydoun, and B. Mccusker, "Big data management and analytics metamodel for IoT-enabled smart buildings," *IEEE Access*, vol. 8, pp. 169740–169758, 2020.

[82] *Browse Azure Architectures—Azure Architecture Center | Microsoft Learn*. Accessed: Aug. 7, 2023. [Online]. Available: https://learn.microsoft.com/en-us/azure/architecture/browse/
[83] *IoT | AWS Architecture Center*. Accessed: Aug. 7, 2022. [Online]. Available: https://aws.amazon.com/architecture/iot/
[84] S. Kotusev. *A Comparison of the Top Four Enterprise-Architecture Methodologies*. Accessed: Oct. 13, 2023. [Online]. Available: https://www.bcs.org/articles-opinion-and-research/a-comparison-of-the-top-four-enterprise-architecture-frameworks/
[85] L. Chamari, P. Pauwels, E. Petrova, E. Chochanova, R. Sebastian, N. Mutsaers, B. Veldhuis, S. Hopkins, N. de Jong, J. van der Velden, J. W. Dubbeldam, and J. van der Weijden. (2021). *Deliverable D4.03 Study of Data Needs and Requirements in Smart Buildings*. [Online]. Available: https://brains4buildings.org/wp-content/uploads/2022/05/B4B-WP4-D4.3_Data-Needs-and-Requirements.pdf
[86] *Cloud Native, Open Source, High-performance Messaging*. Accessed: Aug. 10, 2023. [Online]. Available: https://nats.io/
[87] L. Chamari, E. Petrova, and P. Pauwels, "Extensible real-time data acquisition and management for IoT enabled smart buildings," in *Proc. Eur. Conf. Comput. Construct.*, Heraklion, Greece, Jul. 2023, doi: 10.1109/ICHR.2006.321337.
[88] *IFC.js*. Accessed: Aug. 28, 2023. [Online]. Available: https://ifcjs.io/
[89] Home | xeokit-convert. [Online]. Available: https://xeokit.github.io/xeokit-convert/docs/
[90] *IFCtoRDF*. Accessed: Aug. 7, 2023. [Online]. Available: https://github.com/pipauwel/IFCtoRDF
[91] L. Chamari, J. V. D. Weijden, L. Boonstra, and S. Hoekstra, "metadata schema generation for data-driven smart buildings," in *Proc. 11th Linked Data Archit. Construct. Workshop*, Matera, Italy, Jun. 2023, pp. 136–147.
[92] S. van Gool, D. Yang, and P. Pauwels, "Integrating sensor- and building data flows: A case study of the IEQ of an office building in The Netherlands," in *Proc. ECPPM*, 2021, pp. 328–333.
[93] G. Desogus, E. Quaquero, G. Rubiu, G. Gatto, and C. Perra, "BIM and IoT sensors integration: A framework for consumption and indoor conditions data monitoring of existing buildings," *Sustainability*, vol. 13, no. 8, p. 4496, Apr. 2021.
[94] N. Moretti, X. Xie, J. Merino, J. Brazauskas, and A. K. Parlikad, "An openBIM approach to IoT integration with incomplete as-built data," *Appl. Sci.*, vol. 10, no. 22, pp. 1–17, Nov. 2020.
[95] H. Chen, L. Hou, G. Zhang, and S. Moon, "Development of BIM, IoT and AR/VR technologies for fire safety and upskilling," *Autom. Construct.*, vol. 125, May 2021, Art. no. 103631.
[96] C. Quinn, A. Z. Shabestari, T. Misic, S. Gilani, M. Litoiu, and J. J. McArthur, "Building automation system–BIM integration using a linked data structure," *Autom. Construct.*, vol. 118, Oct. 2020, Art. no. 103257.
[97] J. Teizer, M. Wolf, O. Golovina, M. Perschewski, M. Propach, M. Neges, and M. König, "Internet of Things (IoT) for integrating environmental and localization data in building information modeling (BIM)," *Proc. 34th Int. Symp. Autom. Robot. Construct.*, Jul. 2017, pp. 603–609.
[98] P. Penna, G. L. Regis, A. Schweigkofler, C. Marcher, and D. Matt, "From sensors to BIM: Monitoring comfort conditions of social housing with the klimakit model," in *Proc. Int. Conf. Cooperat. Design, Vis. Eng.*, in Lecture Notes in Computer Science: Including Subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics, vol. 11792, 2019, pp. 108–115.
[99] *TheAgentK/Tuya-Mqtt*. Accessed: Aug. 7, 2023. [Online]. Available: https://github.com/TheAgentK/tuya-mqtt
[100] J. Koh, D. Hong, R. Gupta, K. Whitehouse, H. Wang, and Y. Agarwal, "Plaster," in *Proc. 5th Conf. Syst. Built Environ.* New York, NY, USA: ACM, Nov. 2018, pp. 1–10.
[101] G. Fierro, A. Saha, T. Shapinsky, M. Steen, and H. Eslinger, "Application-driven creation of building metadata models with semantic sufficiency," in *Proc. 9th ACM Int. Conf. Syst. Energy-Efficient Buildings, Cities, Transp.*, New York, NY, USA, Nov. 2022, pp. 228–237.
[102] *Event-Driven Architecture*. Accessed: Aug. 7, 2023. [Online]. Available: https://aws.amazon.com/event-driven-architecture/

LASITHA CHAMARI received the B.Sc. degree in engineering (electrical and information engineering) from the University of Ruhuna, Sri Lanka, in 2014, and the M.Sc. degree in electrical installations from the University of Moratuwa, Sri Lanka, in 2020. She is currently pursuing the Ph.D. degree with the Information Systems in the Built Environment Research Group, Department of the Built Environment, Eindhoven University of Technology. She is currently working on the Brains4Buildings Project in the Netherlands. Her current research interests include developing methods for the integration of heterogeneous data for data-driven building applications, the IoT, smart buildings, building information modeling, semantic web technologies, and digital twins.

EKATERINA PETROVA received the Ph.D. degree in civil engineering from Aalborg University, Denmark, in 2019. She was a Visiting Researcher with the Department of Architecture and Urban Planning, Ghent University, in 2018. She joined the Information Systems in the Built Environment Research Group, Eindhoven University of Technology, in March 2021. She is currently an Assistant Professor of artificial intelligence in construction with the Department of the Built Environment, Eindhoven University of Technology. Her current research interests include the integration of various symbolic and statistical artificial intelligence approaches for decision support in performance-oriented building design and engineering, artificial intelligence in the built environment, ambient intelligence, cognitive approaches, smart buildings, building information modeling and management, and semantic web technologies. She has been working on various topics related to the implementation of symbolic and statistical artificial intelligence approaches for decision support in the context of sustainable building design, circular buildings, digital twins, and data-driven smart buildings.

PIETER PAUWELS is currently an Associate Professor with the Department of the Built Environment, Eindhoven University of Technology. Previously, he was with the Department of Architecture and Urban Planning, Ghent University, from 2008 to 2019. With a lot of experience and knowledge in computer science and software development, he is involved in several industry-oriented research projects on topics affiliated with AI in construction, design thinking, building information modeling (BIM), linked building data (LBD), linked data in architecture and construction (LDAC), and semantic web technologies. His current research interests include information system support for the building life-cycle, such as architectural design, construction, and building operation.