Title

Enhancing Building's Operation and Maintenance with Digital Twin technology: A Predictive Maintenance of Linear Motor Elevators

A thesis presented for the degree of Master of Science in Information Systems

Student ID No. / Name

23148 AZIZ BIO GBEDOUROROU

Supervisor

Professor Sandor Markon

Date of Submission

September 13, 2025

神 戸 情 報 大 学 院 大 学

情報技術研究科 情報システム専攻

Kobe Institute of Computing, Graduate School of Information Technology

Abstract

The rapid growth of urban areas is drastically changing construction and building management. Estimates by the United Nations place the world's population in 2050 at 10 billion, with 70% of these people expected to reside in urban areas compared to 55% today [1]. This demographic shift implies that over 200,000 people are forecasted to migrate to urban areas daily, and construction of over 13,600 multifamily units and more than 3,600 non-residential buildings per day is required to accommodate increased demand [1]. Interestingly, African urban areas have been found to have a 3.1% annual growth rate, which corresponds to a requirement for construction of more than 500 buildings per day to achieve infrastructure as well as residential demand [1].

However, a considerable percentage of these infrastructures, especially in areas of lower development, fail to observe proper maintenance procedures. Consequently, these suffer from shorter machinery life, functional breakdowns, and increased safety risks. This research addresses these concerns by using Digital Twin (Digital Twins) technology to enhance the O&M procedures of building equipment by integrating real-time data tracking and visualization.

The research method involves developing a browser-based Digital Twin platform using TypeScript and Three.js, that utilize data captured on various sensors. Real-time data from sensors are managed using a Unified Namespace structure based on MQTT-based protocols and thereafter represented using dashboards. The results expected include a reduction in elevator inactive time, integration of proactive maintenance techniques, and increased asset life. This research promotes and illustrates the importance and prospect of using Digital Twin techniques for predictive maintenance purposes on building facilities, particularly for LMEs.

Table of content

List of Abbreviations

AI - Artificial Intelligence

AWS - Amazon Web Services

Digital Twins - Digital Twin

HVAC - Heating, Ventilation, and Air Conditioning

IoT - Internet of Things

ML - Machine Learning

MQTT - Message Queuing Telemetry Transport

UNS - Unified Namespace

VR - Virtual Reality

LM - Linear Motor

MAX - TK elevator's predictive maintenance platform

MULTI - TK elevator's ropeless elevator system

O&M - Operation and Maintenance

TK - TK elevator (formerly ThyssenKrupp)

ADXL343 - 3-axis Digital Accelerometer sensor

DHT11 - Digital Temperature and Humidity sensor

IC - Integrated Circuit (in context of IC Recorders)

MPU6050 - 6-axis Motion Processing Unit sensor

API - Application Programming Interface

Three.js - JavaScript 3D library

URL - Uniform Resource Locator

NASA - National Aeronautics and Space Administration

UN - United Nations

CO2 - Carbon Dioxide

GIS - Geographic Information System

List of figures

List of tables

Chapter 1. Introduction

1.1 Background

Today, the world population is undergoing a demographic shift. According to the UN, by 2050, the world population could reach 10 billion. With 68–70% living in urban areas, up from 55% in 2018 [2]. Over 200,000 people will move into urban areas, putting immense pressure on infrastructure such as roads, public transit, housing stock, and basic public services. To meet the needs created by this rapid urban growth, urban areas will need to build over 13,600 multifamily units and more than 3,600 non-residential buildings per day [1]. In Africa alone the construction of 500 new buildings daily is required to meet current demand [1].

The growth process poses a major challenge: ensuring sustainable O&M of infrastructure in buildings, especially with respect to vertical mobility systems, such as elevators. Traditional maintenance approaches, generally classified as either corrective or planned, are insufficient to meet safety and reliability demands of high-density urban buildings. In the context of Sub-Saharan Africa, where the urbanization rate is over 3.1% annually, technical competence deficiencies, poor management, and outdated practices further exacerbate these challenges.

On the other hand, modern technological advancements such as LMEs are an important innovation in vertical transportation systems in buildings. Unlike conventional roped elevators, LMEs operate on magnetic linear propulsion systems, thus dispensing with cables. The architecture of the elevators supports multi-directional motion, high operating speeds, and the simultaneous transportation of various elevator cars within one shaft, attributes that make them particularly well-suited to super-high-rise buildings and high-throughput environments [3]. However, the complexity involved in LMEs, including large-scale electromechanical configurations, complex control systems, and built-in magnetic components, makes traditional O&M practices mostly infeasible. As put forward by Onat et al. [3], “linear synchronous motors. can be considered as core components of a new generation of elevator systems.”

To address these challenges, the construction and facility management industries are turning to Digital Twin (Digital Twins) technology. A Digital Twin is a dynamic, real-time digital representation of a physical asset that integrates sensor data, 3D visualization, and machine learning to enable continuous monitoring and predictive maintenance [4]. Applied to LMEs, Digital Twins allow real-time fault detection, performance simulation, and automated diagnostics—capabilities unattainable through manual inspections or periodic servicing [5].

Leading companies have already begun implementation. TK elevator’s MULTI system, the world’s first commercial ropeless elevator, uses linear induction motors to support multi-car, horizontal-vertical mobility, reducing elevator footprint by 50% and increasing throughput by 50% [6]. Paired with Digital Twins-based platforms such as MAX, predictive maintenance is now deployed at scale, improving reliability and reducing downtime by up to 50% [7].

This research lies at the intersection of urban development, advanced elevator systems, and predictive maintenance. It explores how Digital Twin technology can be integrated into Linear Motor elevator systems to enhance performance, ensure safety, and extend asset lifespan—especially in rapidly urbanizing, resource-constrained environments such as those in Africa. By doing so, it contributes to the growing body of knowledge on intelligent building systems and offers a scalable model for sustainable urban infrastructure.

1.2 Problem analysis

The Operation and Maintenance phase is a critical phase of a building life cycle. It has significant impacts on its sustainability, functionality, and safety. Poor maintenance practices, however, are widespread globally, and the resulting challenges are especially acute in developing countries and complex infrastructure systems [8] [9].

1.2.1 Inefficient maintenance practices as a core issue

This research primarily focuses on how poor maintenance practices are widely adopted in the context of infrastructure development. Conventional maintenance methodologies are mostly reactive or preventive and do not satisfy to address failure prediction and innovate resource allocation, especially when it comes to multi-dimensional inherencies of complex and dynamic systems (i.e. elevators).

This inefficiency has several adverse consequences:

- Increased Operation and Maintenance expenses due to excessive emergency repairs unplanned outages.
- Decreased equipment life (i.e. equipment fails faster without proper predictive maintenance).
- Frequent outages and interruption to operations, affecting building usability and occupant experience.
- Increased environmental impact, as ineffective maintenance practices will lead to the waste of both resources and waste.
- Increased concerns for accidents and being non-compliant with regulations, safety risks and unclear potential liabilities.

1.2.2 Challenges in Developing Countries

Developing countries, including many countries in Africa, face barriers, which intensifies the issue of poor maintenance

Mismanagement of public resources: Poor allocation or monitoring of resources diminishes the possibility of completing routine maintenance.

Insufficient laws and regulations: The lack of, or insufficient, standards and codes of maintenance increase poor facility management [10].

Lack of knowledge and awareness: Stakeholders do not have awareness of the best practices or technologies that could be used.

Lack of training and proper tools: Maintenance personnel often continue using antiquated methodologies

without access to modern data-approved approaches.

Poor maintenance culture: Reactive maintenance remains entrenched regardless of available financial resources and other priority constraints [8].

1.2.3 Challenges in Developed Nations

Even though developed countries have better organized maintenance systems, they are still facing some challenges. Mainly coming from the complexity of modern building systems and the strict requirements of regulatory authorities.

Complexity of building systems: Highly sophisticated facilities, including LMEs, have complex components and ever-changing operating characteristics. These evolving challenges cannot be managed without forensic inspections based on prior experience [5].

Strict regulatory compliance: Safety and efficiency standards change due to regulatory agencies and continuous documentation; tracking and reporting requirements prohibit the use of traditional maintenance strategies.

Energy efficiency requirements: Sustainability and a higher demand for sustainability forces facilities to reduce energy use, which only complicates the maintenance demands.

Lack of labor: The high level of specialization needed to manage complex systems limits the labor pool, thereby emphasizing the need to design automated and intelligent solutions into maintenance strategies [4].

Figure 1 shows the problem tree of this research.

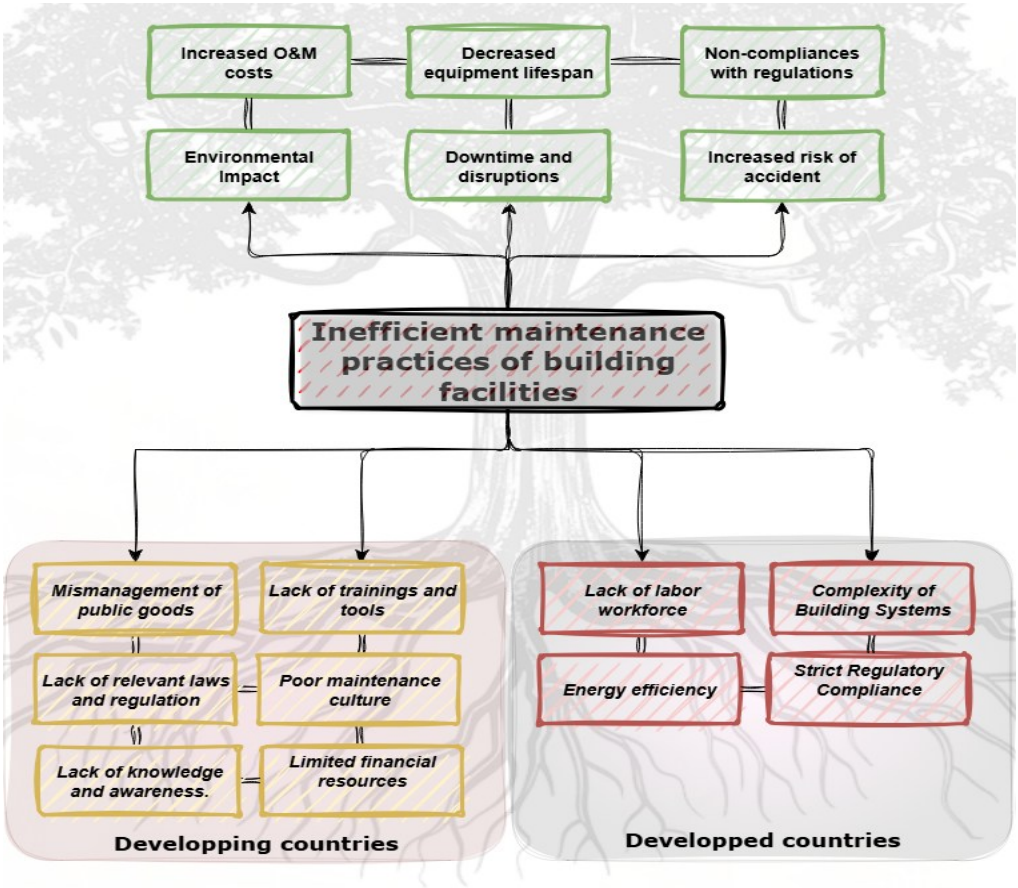


Figure 1: Problem tree

1.3 Issue to be addressed

In the case of ineffective maintenance strategies, this research narrow focuses on the maintenance and operational issues associated with a new class of elevator system called a LME, one of the most advanced and geometrically complicated elevator systems available.

What makes an LME fundamentally different from conventional elevators are its ropeless design, its magnetic linear propulsion system, and its multi-dimensional movement capacity [3]. These attributes lend themselves well to buildings with demand for vertical transportation in high-rise and ultra-high-rise buildings, providing superior spatial capabilities, energy efficiency, and transport capacity. However, such advanced attributes generate challenges in maintenance, which typically has not been documented before:

- Structural and Operational Complexity: The distributed design of LMEs has elaborated mechanical systems, embedding electronics, and electromagnetic subsystems, which eliminates the practicality of maintaining the LMEs based on inspections, which is a common practice used in conventional elevators [3].
- Data Richness: LMEs generate continuous data including large volumes of operational and performance data collected from various onboard sensors. These sensors monitor diverse parameters such as magnetic fields, vibrations, current and temperatures. In addition, the presence of multiple lifts, cabins, and magnets throughout the hoist ways introduces complex data streams from each component of the system. Managing this volume and variety of data requires advanced machine capabilities. Traditional maintenance approaches failed to keep up with such demands, making them unsuitable for LMEs.
- Real-Time Synchronization Requirements: LMEs operate in a multi-directional, high-speed, and multi-vehicle operational context that will impose real-time and precise monitoring and synchronization between LMEs and their management systems, far exceeding the possible effectiveness of manual and systemic inspections [11].

In addressing these fundamental dilemmas, this study conceives the use of Digital Twin technology in LME maintenance practices. Digital Twin systems provide virtual and real-time replicas of physical assets such that operators can monitor system performance, identify anomalies, predict failures, and optimize operations in real-time [4]. With the incorporation of IoT sensors, machine learning algorithms, and advanced analytics, Digital Twin systems can create processes to report maintenance as a predictive rather than a reactive response.

This study will explicitly participate in the creation and validation of a full-cycle predictive maintenance platform based on Digital Twin technology that addresses the specific operational and maintenance profiles of LMEs. With the establishment of this sense of Digital Twins and predictive maintenance, this study hopes to deliver the following:

- Significant decreases in interruptions and elevator downtime.
- Extended equipment longevity and usefulness.
- Increased adherence to stringent compliance requirements.
- Significant decreases in lifecycle maintenance costs.

This study is conducted with a committed Tankyu Practice spirit as it is a strong key for success in solving issues. Tankyu is a research methodology that combines search, quest or inquiry with pursuing research to identify and solve the ultimate problem. The Tankyu chart defines the identified issue, possible solutions, and the enablers (business model, technologies and human resources) to achieve the solution [11]. Figure 2 shows the Tankyu chart for our research.

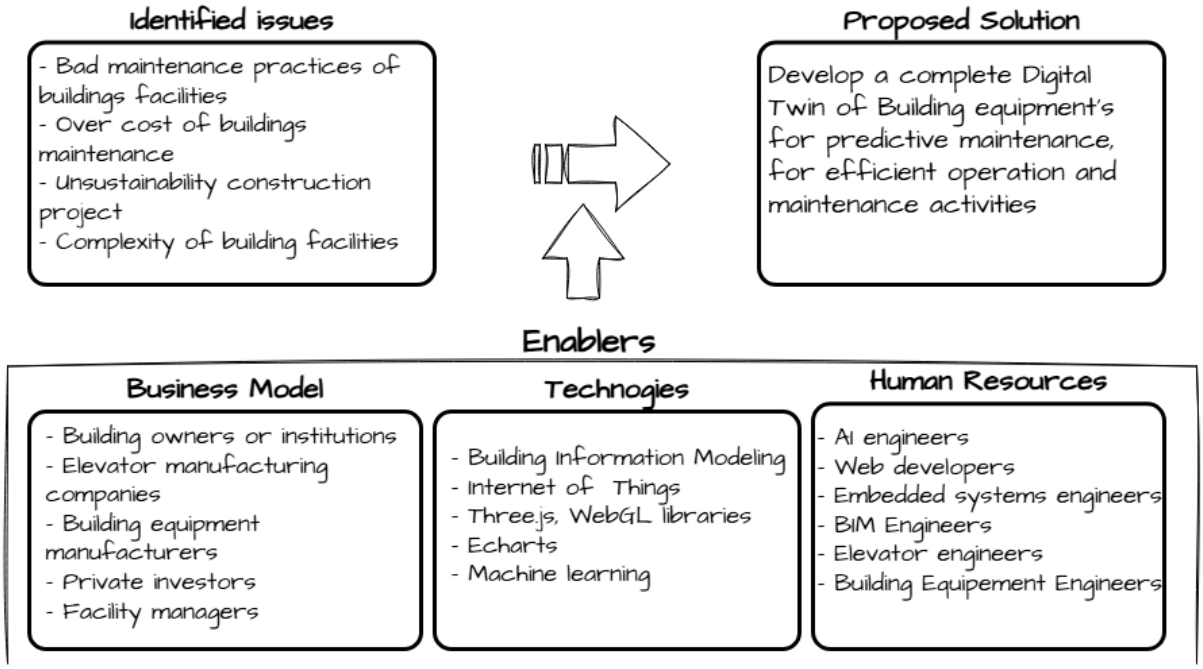


Figure 2: Tankyu Chart

1.4 Document structure

This thesis is composed of six principal chapters, which are organized to sequentially address the research objectives, applied methodology, design solutions, validation, and repercussions for the use of Digital Twin technology for predictive maintenance of LMEs.

Each chapter is outlined.

Chapter 1: Introduction - The chapter introduces the rapid urbanization context, issues with Operation and Maintenance, LMEs and Digital Twin context, a complete problem analysis, indicating which specific problem, and summary of the structure of the thesis.

Chapter 2: Literature Review - This chapter is a review of existing scholarly works and industry developments in elevator maintenance, LME technologies, Digital Twin frameworks and predictive maintenance. This chapter identifies research gaps and positions the thesis against recent state-of-the-art materials.

Chapter 3: Methodology - In this chapter the research methodology is detailed, including the technical architecture of the designed Digital Twin platform and its deployed sensors, data management techniques and setup.

Chapter 4: Solution design and prototype development - This chapter describes the practical development

of the Digital Twin platform specifically designed for LMEs. It discusses the prototype features, including the 3D views, interactive dashboards, heatmaps, animation tools, and the ingestion of live sensor data streams.

Chapter 5: Verification and results - This chapter evaluates the developed Digital Twin prototype performance. It describes the tests, simulations, and analyses conducted to validate the effectiveness of the predictive maintenance approach. The findings include improvements in fault prediction accuracy, operation effectiveness, and system reliability.

Chapter 6: Conclusion - This Chapter summarizes the key research contributions, discusses practical implications and the limitations of the work, then concludes with recommendations for future research as well as further applications to other complex building systems.

The thesis includes an Appendix with a variety of supplementary details such as diagrams, sensor log data and code snippets, as well as a complete References section, citing all the relevant literature and resources consulted.

Chapter 2. Literature Review

2.1 Introduction

This literature review intends to gather a complete outline of previous research conducted on Digital Twins technology, predictive maintenance techniques, and applications in building Operation and Maintenance, specifically directed towards LMEs. We outline the process to evaluate related literature and synthesize applicable findings. The literature review is meant to be a foundation of theory for future research on improving buildings Operation and Maintenance with Digital Twins technology.

The literature review includes three main areas: Digital Twins technology, predictive maintenance techniques, and LMEs. We will describe the definitions, the history of Digital Twins, and applications of Digital Twins in various fields but will also focus on their application in building Operation and Maintenance. This section will also review the current literature about LMEs specifically related to Operation and Maintenance.

The literature discussed above directly informs this study's research question and research objectives. The research question is as follows: In what ways can Digital Twin technology be used to improve the Operation and Maintenance strategies for LMEs through predictive maintenance approaches? To address the research question, the literature review will provide an overview of the current state-of-knowledge surrounding Digital Twins, predictive maintenance, and LMEs to determine gaps and possibilities that the study will address.

Furthermore, the literature review will provide support for the research objectives:

- To explore the perceived advantages and disadvantages of utilizing Digital Twin technology for building Operation and Maintenance, related specifically to LMEs.
- To examine the use of predictive maintenance techniques associated with Digital Twin technology to ensure the performance and reliability of LMEs.
- To develop and articulate a framework of Digital Twin-based predictive maintenance for LMEs that affords the characteristics and needs of the systems.

2.2 Overview of Digital Twin Technology

2.2.1 Definition and Core Concepts

Digital Twin technology signifies a fundamental change in our understanding of and engagement with physical things in the digital domain [12]. Digital Twins, Internet of Things, blockchains, and Artificial Intelligence could redefine our imagination and our view of globalization. A Digital Twin is, at its most basic, a digital representation of something, objects, people, activities and either existing or intended. A Digital Twin understands the life cycle of the object and is continuously updated with real-time data. It incorporates simulation, machine learning, and reasoning, and assists us in making decisions (see Figure 3).

The concept goes beyond just digital modeling or simulation. If you picture traditional digital modeling as

a static you, or example that models your specifications, a Digital Twin produces a dynamic twin that will always have a relationship and continuous correspondence with the physical thing that it is replicated [13]. The two-way data flow of a Digital Twin is the key factor that sets it apart from simulation models. Digital Twins provide a data-rich approach to simulating real-life scenarios with the use of real-time data, simulation, machine learning, and reasoning to show potential future scenarios and help organizations make decisions.

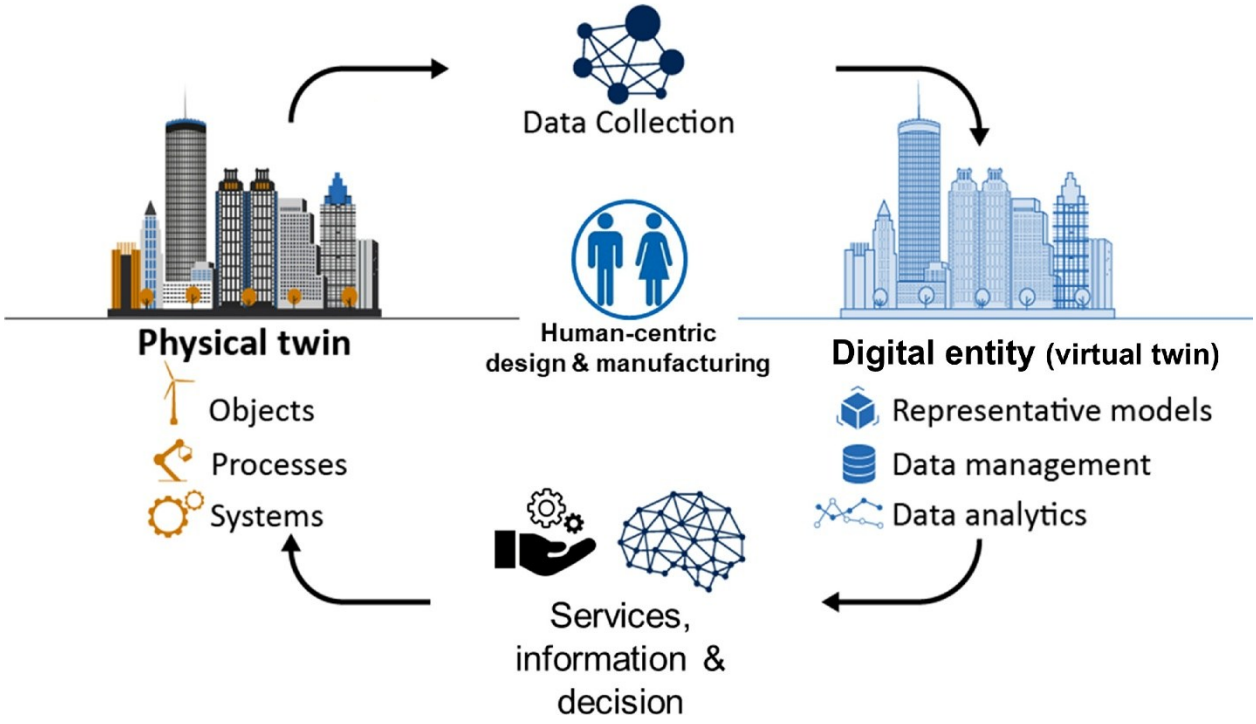


Figure 3: Digital Twin flow; [14]

2.2.2 Historical Development and Evolution

The origins of Digital Twins date back to the 1960's, at NASA. As NASA began developing the Apollo missions, it created and built high-fidelity physical simulators [14] — almost replicas of the spacecraft — to allow astronauts, engineers and others to practice not only activities, but also emergency procedures. During the Apollo 13 in-flight incident in 1970, NASA took advantage of the simulators, partially recreating the in-flight problem as well as testing the acquired solutions and ultimately brought the crew home safely (see Figure 4) [15].

In the following decades, the original concept of Digital Twin continued to evolve. In 2002, Dr. Michael Grieves initially introduced the notion of the Digital Twin for manufacturing as a 'digital' version of a physical product or item. In the years following, NASA's John Vickers is credited with bringing the name of Digital Twin to recognition in 2010 [16].

What has changed in those two short decades? Technology. Several technologies came onto the scene such as the Internet of Things (IoT), cloud computing, and advanced analytical data systems that permitted devices to continuously stream data in real-time through cloud environments. These advances in technology

have fundamentally changed how we communicate and utilize Digital Twins. Digital Twins have become distinct from simulations and experiments and are now considered live digital counterparts that support the ability of industry to monitor, predict, and enhance the performance of physical systems.

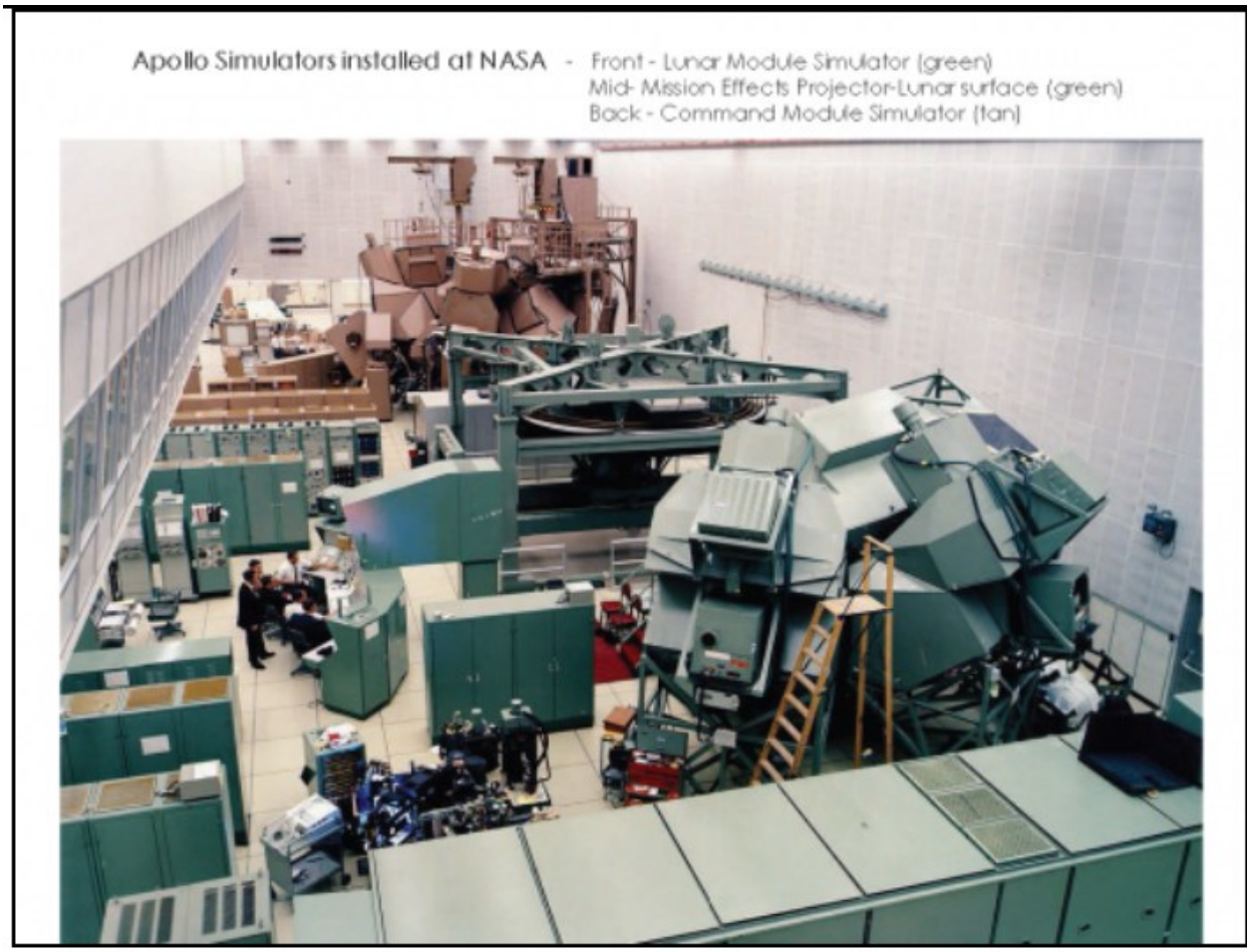


Figure 4: The Apollo simulators (Digital Twin origin); [16]

2.2.3 Benefits and Value Proposition

Digital Twin technology offers numerous benefits across various dimensions of operation:

Operational Efficiency

Digital Twins significantly increase operational efficiency by offering real-time insights into physical assets and operations. Organizations can monitor performance in real-time, identify bottlenecks, and optimize processes based on data-driven insights.

Predictive Maintenance

Digital Twins can help anticipate possible faults by modeling system performance. This allows for proactive maintenance to save downtime and boost efficiency. By analyzing patterns and anomalies in real-time data, Digital Twins can predict when equipment is likely to fail, enabling maintenance teams to intervene before costly breakdowns occur.

Cost Reduction

By streamlining resource management and cutting waste, Digital Twins' predictive powers result in significant cost savings. Organizations report substantial reductions in maintenance costs, improved asset utilization, and decreased downtime.

Enhanced Product Development

Digital Twins provide a risk-free product development environment, allowing design and engineering teams to explore more design options without the cost associated with the production and testing of physical prototypes [17]. Companies can reduce expenses and development cycles by testing new concepts in Digital Twins before they are physically created.

2.2.4 Applications Across Industries

The versatility of Digital Twin technology has led to its adoption across diverse industry sectors:

Manufacturing

Digital Twins are most widely used in the manufacturing industry [18]. Manufacturing relies on high-cost equipment that generates a high volume of data, which facilitates creating Digital Twins. Most importantly, these Digital Twin applications have allowed for substantial cost savings in manufacturing processes extensively.

Healthcare

Digital Twin in healthcare can improve the design, development, testing, and monitoring of new drugs and medical devices. Digital Twins in healthcare represent a major transformation in how medical data is utilized. They enable the creation of virtual replicas of patients, medical devices, healthcare systems, and even entire hospitals [20].

Automotive and Transportation

The automotive industry is at the forefront of embracing Digital Twin applications, leveraging it to enhance design, manufacturing and maintenance [18]. In the automotive sector, Digital Twins streamline testing and support condition-based maintenance.

Aerospace

Building on its origins, the aerospace industry continues to be a leader in Digital Twin adoption. 75% of air force executives have expressed confidence in the Digital Twin [19]. The technology is used for everything from design optimization to real-time monitoring of aircraft systems.

Energy and Utilities

Digital Twins can reduce a building's carbon emissions by 50%, significantly enhancing sustainability efforts. In the energy sector, Digital Twins optimize the performance of power plants, manage renewable energy sources, and enhance grid reliability.

Smart Cities and Infrastructure

Singapore completed a Digital Twin of its entire country in 2022, which incorporates AI, lidar, 3D mapping, raw GIS and imagery data, and other metaverse technologies to allow the country to develop more resilient, sustainable, and smart infrastructure [21].

2.3 Digital Twins in Building's Operation and Maintenance

2.3.1 Introduction to Building Digital Twins

Digital Twins for building Operation and Maintenance offer a paradigm shift in the way we arrange and manage facilities, advancing from a static view of building systems to a dynamic live replica of building systems, allowing real-time monitoring and analysis, and optimization [22]. Building Digital Twin systems differs from traditional Building Management Systems (BMS) where the BMS is only concerned with collecting data and limited functions of control, building Digital Twins view data from dozens of sources, provide and serve analytical capability and simulation [9].

The development of building Digital Twins has progressed due to the convergence of all sorts of different technologies varying from the expanse of Internet of Things (IoT) sensors, the power and availability of cloud computing infrastructures, and the advanced levels of machine learning algorithms. These technologies bring together a multitude of interconnected devices capable of creating a sophisticated virtual building model which utilizes the flow of data and 'predicts' system performance, optimizes energy and consumption, diagnoses issues, and anticipates the need for maintenance proactively, before a failure occurs.

2.3.2 Key Applications in Building Systems

2.3.2.1 HVAC System Optimization

Heating, Ventilation, and Air Conditioning (HVAC) systems are one of the leading areas of Digital Twin within buildings. Digital Twins can optimize HVAC operations in real-time by continuously measuring environmental factors, occupancy, and performance metrics for the system [23]. In their research [24], found that Digital Twins of HVAC systems were able to provide utility savings of as much as 25% without sacrificing comfort using predictive control algorithms that could trained to predict thermal loads and change how the system operates.

Typically, the integration of an HVAC Digital Twin usually refers to the use of many different sensors (temperature, humidity, CO2, occupancy, etc.) that provide data to the Digital Twin as a continuous stream. This data is first analyzed by advanced machine learning algorithms to detect patterns, discuss future possibilities, and ultimately derive optimized responses for the system during operation.

2.3.2.2 Energy Management and Efficiency

Building energy management is another vital application area in which Digital Twins add great value. Energy management systems enabled by Digital Twins can achieve reductions in overall building energy consumption through intelligent load balancing, integration of renewable generation, and optimized demand response [22].

For instance, [25] found that a commercial office-based building, with the implementation of a comprehensive energy Digital Twin, reduced their overall energy costs by 30% and improved their energy from renewable energy use efficiency by 40%. The Digital Twin achieved these reductions because it continually analyzed how the building consumed energy, predicted when future electrical demand would

occur, and automatically adjusted the building systems to optimize energy usage while maintaining occupant comfort.

2.3.2.3 Structural Health Monitoring

Digital Twins are being used in greater applications of structural health monitoring, helping facility managers to continuously assess the integrity of a building and determine if there are any potential structural issues emerging beforehand [26]. Sensor networks developed to monitor information and data on structural parameters such as vibration, strain, temperature differences, and settlements and transfers this information to the Digital Twin models to detect anomalies and prepared for future structure behavior based on the assumed condition of loading.

A notable case study [27] presented a structural digital twin developed for a high-rise building. In this case, the digital twin enabled engineers to monitor foundation settlement without making assumptions. It could determine whether one or both parts of the foundation were settling and assess whether the settlement could potentially lead to significant structural damage.

2.3.3 Technologies to develop Digital Twins

The development of Digital Twin platforms requires two fundamental technological components: the visual twin and the data twin. The visual twin encompasses the tools and frameworks necessary to create and display three-dimensional representations of physical assets, while the data twin provides the communication infrastructure that enables real-time data exchange between physical systems and their digital counterparts.

2.3.3.1 The Visual Twin

The visual twin represents the graphical and interactive aspects of Digital Twin systems, requiring both modeling tools for asset creation and web-based viewers for real-time visualization.

3D Modeling Tools

Creating accurate digital representations of building systems requires specialized modeling software capable of handling complex geometries and system relationships. Several categories of tools serve this purpose in Digital Twin development.

2.3.3.1.1 Building Information Modeling (BIM) Platforms

Traditional BIM software such as Autodesk Revit, Bentley MicroStation, and similar platforms provides the foundation for creating detailed building models. These tools excel at capturing architectural, structural, and mechanical system details with precision. For elevator systems, BIM platforms can model shaft geometry, mechanical components, and spatial relationships between system elements.

Autodesk has also developed Fusion 360 and other cloud-based modeling tools that support collaborative development and version control. These platforms enable the creation of detailed component models that can be exported to various formats for use in Digital Twin applications.

2.3.3.1.2 Web-based 3D Visualization

The visualization component of Digital Twin systems has increasingly moved toward web-based solutions that provide accessibility without requiring specialized software installations.

Three.js

Three.js is a JavaScript library based on WebGL technology that can be used in Digital Twin applications. It provides 3D rendering capabilities directly in web browsers. The library supports loading various 3D model formats, real-time lighting and materials, animation systems, and physics.

Autodesk, a major player in the construction software industry, has developed Autodesk Tandem (see Figure 5). Autodesk tandem is a Digital Twin platform at the building level which is designed mainly for building asset management and monitoring. The software is a cloud-based solution which is built on top of the Autodesk Platform Software (APS) Viewer. The APS viewer, originally called Forge Viewer, is based on Three.js.

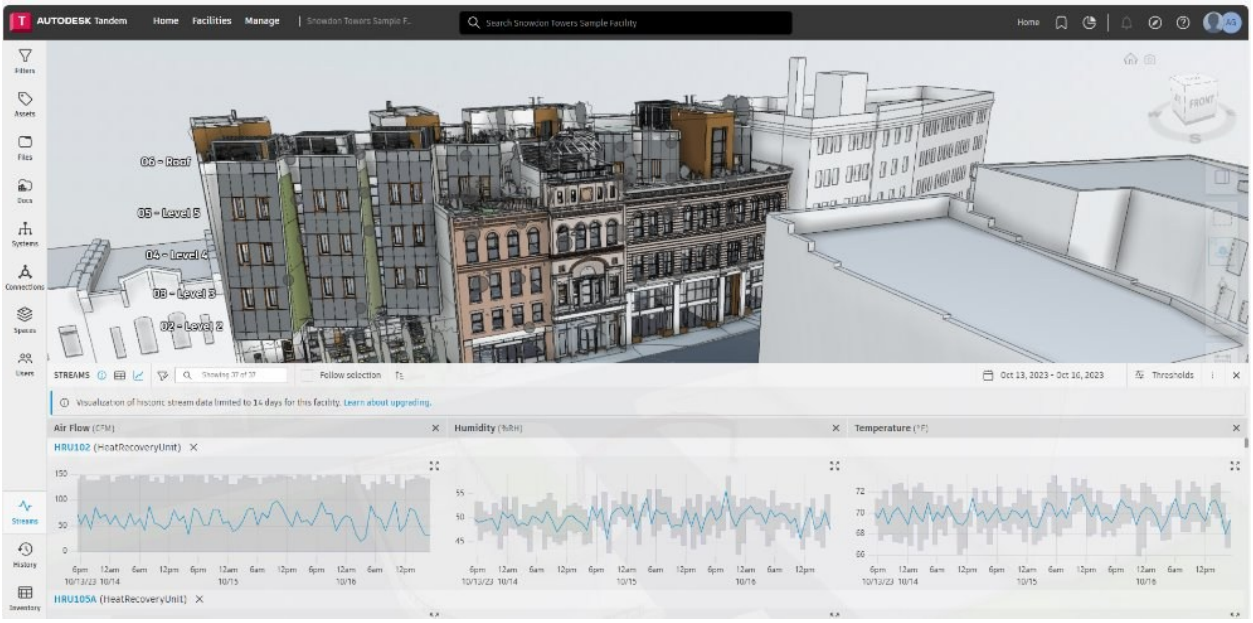


Figure 5: Autodesk Tandem Interface

2.3.3.2 The Data Twin

The data twin encompasses the communication infrastructure and data management systems that enable real-time synchronization between physical assets and their digital representations.

2.3.3.2.1 Unified Namespace (UNS) Architecture

The Unified Namespace concept is an architecture that centralizes operational data from distributed systems. Rather than point-to-point connections between individual systems, UNS creates a single data source of truth that all the applications can access (see Figure 6).

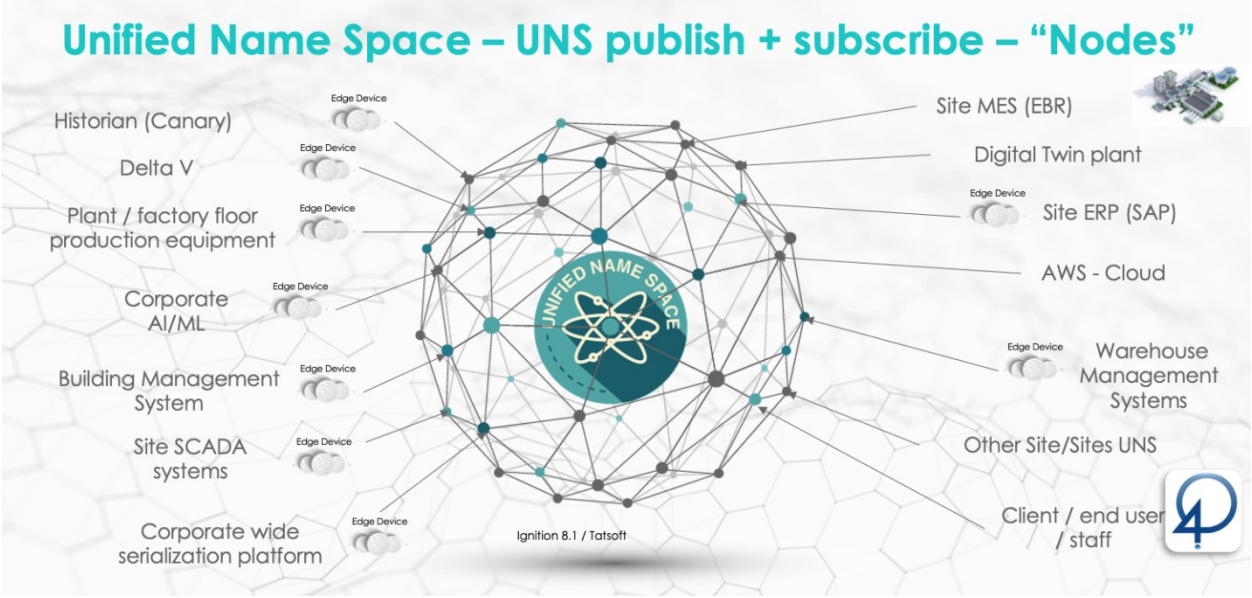


Figure 6: Unified namespace by IIOT 4.0 solutions; [30]

Conceptual Framework

UNS organizes data in a hierarchy and usually follows industry standards such as ISA-95 establishing levels from the enterprise level down to devices. This hierarchy is useful to logically organize complex systems such as buildings where data comes from HVAC systems, elevator controls, lighting systems, and many other data sources.

The use of namespaces allows applications to access data without physically connecting to systems. This abstraction should enable better system integration and offer increased ease in the addition of new applications or data sources to existing installations.

Benefits for Building Systems

Although it originates from the manufacturing industry, the UNS can also be applied to building systems. It will provide several advantages over traditional point-to-point integration approaches. New sensors data can be added without any disruption on the existing connections. Thus, allowing data consistency across multiple applications.

For complex systems with distributed parts that generate millions of data for instance the LMEs, the UNS will enable centralized access to data from distributed sensors and control systems throughout the elevator installation while maintaining logical organization of information from different subsystems.

2.3.3.2.2 MQTT Protocol Implementation

Message Queuing Telemetry Transport (MQTT) is the most used communication protocol for UNS in industrial applications. MQTT is a messaging protocol designed for the Internet of Things (IoT). It is intended to be an extremely lightweight publish/subscribe messaging transport ideal for small code footprint remote devices tailoring to conserve minimum bandwidth.

Protocol Characteristics

A central broker manages message routing, enabling loose coupling between system components. The protocol is designed for efficiency with minimal overhead, making it suitable for systems with limited

network bandwidth or processing capacity. MQTT supports quality of service levels that enable applications to specify delivery requirements ranging from best effort to guaranteed delivery (see Figure 7).

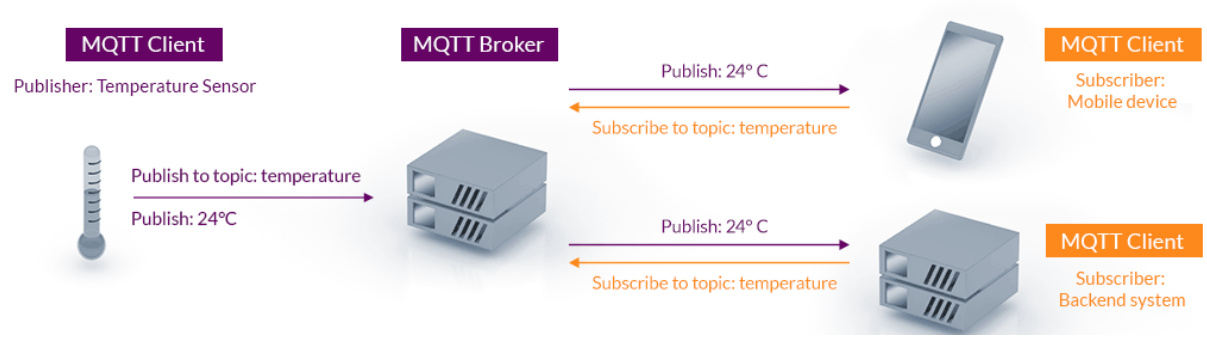


Figure 7: MQTT Protocol; [31]

Topic Hierarchical Structure and Organization

MQTT employs a hierarchical topic structure using forward slashes. Topics can be seen as paths for organizing data in a logical manner. In system-built environments, the topics usually represent the physical structure (or structure elements) of the installation. For example, in a Linear Motor elevator system, you might have the following topics:

building01/elevator01/motor/temperature

building01/elevator01/position/current

building01/elevator01/sensors/vibration

As illustrated, all applications could subscribe to various types of data (all temperature readings) or specific equipment (all data from elevator01). The hierarchical topic structure allows to include wild card subscriptions, which would allow an application to subscribe to building01/elevator01/+ to receive all data from elevator01..

Broker infrastructure

MQTT brokers manage the message route from publishers to subscribers. There are several common implementations, each serving a different deployment:

- Eclipse Mosquitto is an open-source broker implementation commonly used in development and smaller deployments. It implements very basic MQTT functionality while also supporting authentication and SSL encryption.

- HiveMQ offers a community edition and an enterprise or paid edition. The paid edition includes features including clustering, monitoring functions, and security features which are helpful in larger installations.

2.4 Linear Motor Elevators

LMEs represent a significant advancement in vertical transportation technology. Unlike conventional elevators, which rely on centralized traction systems, ropes, and pulleys, LMEs operate using linear motors, distributed electromagnetic propulsion systems that eliminate the need for ropes and enable multi-directional movement within elevator shafts [3] [28].

2.4.1 Comparison Between LMEs and conventional elevators

Table 1 summarizes the primary differences between LMEs and conventional elevators:

Table 1: Comparison between LME and conventional elevator

Feature	Linear Motor elevator	Conventional elevators
Propulsion System	Distributed electromagnetic linear motors along the elevator shaft	Centralized traction motors with ropes and counterweights
Movement Capabilities	Multi-directional (vertical and horizontal), enabling higher transport flexibility	Typically, unidirectional (vertical movement only)
Component Distribution	Components distributed throughout the elevator shaft, embedded directly within structural elements	Components concentrated primarily within a centralized machine room and shaft top
Maintenance Accessibility	Complex due to dispersed components along the shaft, making traditional inspections challenging	Relatively easier with centralized and accessible components
Capacity and Efficiency	Significantly increased transport capacity (up to 50%) and reduced elevator footprint	Limited capacity and efficiency, constrained by rope mechanics and shaft design

The distributed architecture of LMEs, which embeds components such as linear motors, controllers, and sensors along the entire elevator shaft, introduces unique operational and maintenance complexities compared to conventional elevators. This spatial distribution makes traditional maintenance practices, such as manual inspection and routine servicing, practically impossible or highly inefficient due to the difficulty of accessing and monitoring the dispersed components.

2.4.2 Operational Advantages and Challenges of LMEs

LMEs offer numerous operational advantages:

- **Space Optimization:** Reduced space requirements due to elimination of ropes and counterweights.
- **Improved Building Design Flexibility:** Allows novel architectural possibilities with horizontal and vertical movement.
- **Increased Throughput:** Supports multiple elevator cars within a single shaft, significantly enhancing transport efficiency.

However, these advantages come with significant maintenance and operational challenges:

- **Complex Data Management:** Continuous operation generates vast quantities of real-time sensor data, requiring advanced analytics capabilities.
- **Real-time Synchronization Needs:** Precise coordination between physical and digital systems is critical, especially given the elevator's multi-directional and high-speed capabilities.
- **Technical Complexity and Cost:** Higher initial investment and complexity in terms of technology and infrastructure compared to conventional systems.

2.4.3 Current Research and Technological Advances

Research efforts and industrial initiatives in the field of LMEs have primarily focused on addressing operational complexities and enhancing performance through advanced predictive maintenance solutions. Studies by Onat et al. (2010) and So & Chan (2019) highlighted linear motors as fundamental components of next-generation elevator systems, emphasizing the need for innovative control and safety systems tailored specifically to LMEs.

Additionally, prominent industrial implementations, such as TK elevator's MULTI system, demonstrate the practical viability and benefits of LMEs, reducing elevator footprints by half and improving transport efficiency by 50% through innovative magnetic propulsion technology [7].

2.4.4 Digital Twins and Predictive Maintenance in LMEs

Given the distributed and complex nature of LMEs, integrating advanced Digital Twin technology for predictive maintenance becomes crucial. Digital Twins provide real-time virtual models capable of:

- **Continuous monitoring and fault detection**: Instantly identifying and diagnosing performance anomalies through real-time analytics.
- **Proactive maintenance scheduling**: Utilizing machine learning algorithms to predict equipment wear and failures, scheduling maintenance proactively.
- **Operational optimization**: Simulating various scenarios to optimize elevator usage patterns and operational configurations for maximal efficiency and reliability.

Recent studies, such as those by Qiu et al. (2024) and Xu et al. (2022), have demonstrated the successful implementation of Digital Twins in LMEs, highlighting their capability to significantly enhance operational reliability, reduce downtime, and streamline maintenance workflows [29], [5].

2.5 Research Gaps and Opportunities

The literature highlights several research gaps:

Limited comprehensive research explicitly combining Digital Twins technology with predictive maintenance tailored to LMEs.

Insufficient exploration of how predictive analytics and machine learning can optimize multi-directional elevator operations in real-time.

Lack of empirical validation in resource-constrained environments, such as Sub-Saharan Africa, where digital infrastructure and skilled labor may differ significantly from developed regions [22].

Addressing these gaps, this research focuses on developing and validating a Digital Twins-based predictive maintenance model specifically designed for LMEs. By integrating real-time sensor data, advanced analytics, and intuitive visualization platforms, this study aims to contribute substantial empirical evidence to this emerging domain, demonstrating practical feasibility and effectiveness.

2.6 Conclusion

This literature review has established the context, significance, and state-of-the-art advancements related to Digital Twin technology, predictive maintenance methodologies, and Linear Motor elevator systems.

Despite substantial advancements, critical gaps remain, particularly in applying Digital Twins-driven predictive maintenance solutions specifically designed for LMEs within developing and rapidly urbanizing regions. This thesis directly addresses these gaps, providing a comprehensive methodological and practical framework, thereby significantly contributing to the body of knowledge in intelligent maintenance and vertical mobility systems.

Chapter 3. Methodology

3.1 Introduction

This chapter presents the methodology used in the development of a Digital Twin platform for monitoring and analysis of LMEs. Through these pages, we present a detailed explanation of the technical and design choices made along the way of developing the platform, highlighting how data is collected, processed, and represented inside the system. It outlines the core elements of the platform and the technologies behind enabling real-time synchronization between the physical elevator system and its Digital Twin.

3.2 Research Scope and Contributions

This research explores the development of a Digital Twin platform that would monitor LMEs. It focuses on the integration of sensor data, real-time visualization, and system architecture design. Our scope is limited to developing the infrastructure and visualization layer to allow for instant monitoring and analysis of elevator performance. As a middle ground between the physical and mechanical aspects of elevators and sophisticated analytics, the Digital Twin gathers, processes, and displays sensor data in a centralized dashboard based on a 3D web viewer.

Here is the breakdown of the research scope:

- **Digital Twins Platform Development (This work):**
  This work involves designing and implementing the Digital Twins system's backbone, including data acquisition from sensors, real-time synchronization through MQTT protocols, and interactive 3D visualization using Three.js and TypeScript. The platform supports visualization of operational parameters such as temperature and position with an extension framework ready to handle more sensor data.
- **Predictive Maintenance Model (Batu' s Thesis):**
  A separate but related research effort focuses on developing predictive models that utilize the data collected and managed by the Digital Twins platform. These models aim to forecast failures or maintenance needs using AI and statistical techniques applied to historical and live sensor data.
- **Integration:**
  The two research components are designed to work together in synergy. While this thesis gives the infrastructure and real-time data streams, Batu's work builds upon it to obtain and utilize the data to perform predictive analysis with the developed AI model. The Digital Twins platform serves as both a data source and an operational interface for future predictive maintenance needs and simulations.

3.3 System Architecture Overview

The proposed Digital Twin platform is structured around a centralized, broker-based architecture designed to facilitate seamless communication between various system components. At its core lies the **Unified**

Namespace (UNS), an MQTT-based broker system that functions as the single source of truth, enabling all subsystems to publish and subscribe to standardized data streams. This architecture ensures modularity, scalability, and real-time synchronization between the physical environment and the virtual twin.
The architecture is organized into three primary layers (see Figure 8):

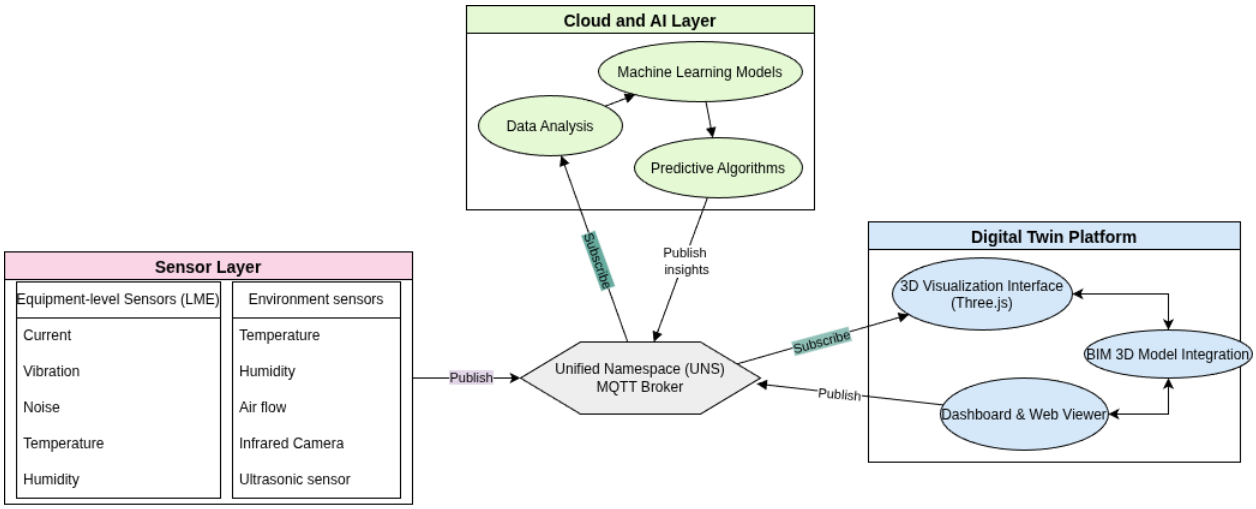


Figure 8: Overall system architecture

3.3.1 Sensor Layer

This layer is composed of all physical sensors deployed on the Linear Motor elevator and in the surrounding environment. These sensors are responsible for capturing critical operational and environmental data and publishing it in real-time to the UNS. They include:

- Equipment-level sensors:
  - Current,
  - Vibration,
  - Noise,
  - Temperature,
  - Humidity
- Environmental Sensors:
  - Temperature,
  - Humidity,
  - Air flow,
  - Infrared Camera,
  - Ultrasonic sensors

The data acquisition process relies on the **MQTT protocol** for lightweight, efficient transmission from edge devices to the broker.

3.3.2 Digital Twin Platform Layer

This layer represents the user-facing system responsible for data visualization, interaction, and simulation. It consists of:

- **3D Visualization Interface**: Built using **Three.js and TypeScript**, the interface presents a real-time, interactive model of the elevator system.
- **BIM 3D model Integration**: Real-world geometric data from elevator components is rendered in the platform.
- **Dashboard:** Users can observe real-time sensor readings and interact with the system through a web interface.

This layer subscribes to live data from the UNS and can also publish control signals or requests back to the broker when needed.

3.3.3 Cloud and AI Layer

While not the focus of this thesis, the cloud layer plays a critical role in the overall system by:

- Subscribing to sensor data from the UNS
- Running **machine learning models** and predictive algorithms developed by a collaborator
- Publishing insights, alerts, or commands back into the UNS

This layer enhances the system's intelligence by enabling proactive maintenance planning and decision support.

3.4 Data Management Strategy

A robust data management strategy is essential for ensuring seamless integration, real-time responsiveness, and data reliability within a Digital Twin system. To facilitate the development and testing of the Digital Twins platform for LMEs, a **two-step data integration approach** was adopted: starting with mock data and transitioning to real-time sensor streams. This phased methodology enabled early-stage feature validation while progressively moving toward operational deployment.

3.4.1 Mock Data Integration

In the initial stage of development, actual sensor hardware was not yet available or fully configured. To address this limitation and accelerate platform prototyping, a dedicated **mock data generator** was developed. This utility was used to simulate realistic operational patterns for:

- **Vibration data** (simulating elevator movement)
- **Temperature and humidity values** (representing environmental conditions)
- **Acoustic signals** (simulating sounds from motors and bearings)

The mock data was designed to mimic expected elevator behavior under both normal and fault-like conditions. These generated datasets were published to the **Unified Namespace (UNS)** via MQTT, using topic structures identical to those planned for real sensors. This ensured that the visualization and interaction modules could be fully developed and tested independently of the hardware layer.

Using mock data enabled:

- Validation of the real-time 3D visualization pipeline
- Early testing of data-driven UI elements (heatmaps, alerts, dashboards)
- Performance benchmarking of the MQTT-to-UI flow

3.4.2 Real-Time Sensor Data Integration

Once sensor installation and calibration were completed, the system transitioned to real-time data acquisition. The sensor suite included:

- **ADXL343 and MPU6050** for position and acceleration
- **DHT11** for temperature and humidity

Each sensor was connected to a Raspberry Pico W microcontroller configured to publish live data over MQTT to the broker installed in a Raspberry Pi 4 using well-defined topic namespaces (e.g., lme/vibration, lme/temperature). The UNS handled all incoming streams and made them available to subscribers, including the frontend visualization interface and optional cloud analytics modules.

This transition was implemented incrementally:

- Individual sensor topics were switched from mock to live sources
- Real-time values were validated against expected patterns
- UI components were updated to reflect and respond to live conditions

This strategy ensured smooth migration from simulation to operation without requiring major code refactoring or reconfiguration.

3.5 Real-Time Synchronization and Platform Design

The Digital Twin platform was designed to enable real-time reflection of physical elevator operations in a virtual environment. This was achieved through a modular architecture, a real-time messaging system using MQTT, and an efficient rendering and update pipeline built with Three.js and TypeScript.

3.5.1 Layered Platform Design

The platform architecture is organized into three core layers:

- **Acquisition Layer:**
  This layer consists of both simulated and real sensors deployed on or near the LME. Devices continuously collect vibration, temperature, humidity, and acoustic data, publishing it to the Unified Namespace.
- **Integration and Management Layer:**
  Powered by an MQTT broker, the UNS facilitates the exchange of messages between producers (sensors) and consumers (visualization components, cloud analytics). This broker acts as a centralized data bus ensuring decoupled communication.
- **Visualization Layer:**
  The frontend, built using Three.js and TypeScript, subscribes to relevant MQTT topics. It interprets sensor data and reflects changes in a 3D model in real-time. Visual indicators include dynamic color coding (heatmaps), object animations, sensor value overlays, and status warnings.

3.5.2 Real-Time Synchronization Mechanism

Figure 9 shows the synchronization between the physical and digital worlds:

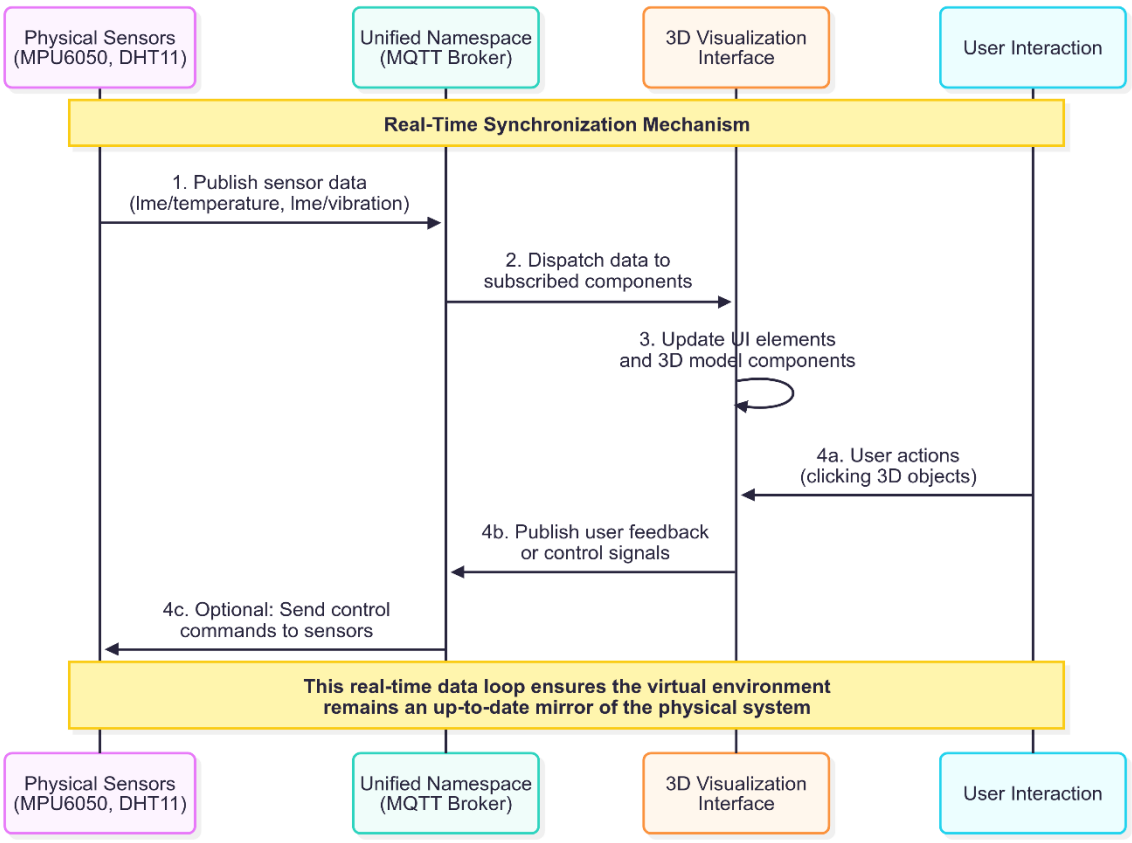


Figure 9: Real-time synchronization interaction diagram

1. **Sensors Publish Data:** Real-time values from physical sensors (e.g., MPU6050, DHT11) are published to specific MQTT topics.
2. **UNS Broker Dispatches Data:** The MQTT broker immediately routes incoming messages to all subscribed components. These include the 3D visualization interface, and the prediction AI models in cloud.
3. **Frontend Reacts to Updates:** The visualization interface listens for specific topics (e.g., ldt/temperature, ldt/humidity) and updates the corresponding UI elements or 3D model components.
4. **User Interaction and Feedback:** User actions (e.g., clicking a 3D object) can trigger requests or feedback into the system, which may also be routed through the broker for logging or control purposes.

This real-time data loop ensures that the virtual environment remains an up-to-date mirror of the physical system, that allows for timely decision-making and real-time monitoring.

3.5.3 Tools and Technologies

Our Digital Twin platform is developed with a modern web development stack fully optimized for performance, modularity, and real-time responsiveness.

Table 2 shows the tools and technologies used in this research.

Table 2: Tools and Technologies used

	Technology	Category	Purpose
	Autodesk Fusion 360	3D modeling	Design from 2D Sketches to the 3D modeling of the Linear Motor System
	Blender	3D modeling	Processing and Export to free format .glb format supported in Three.js
	Three.js	3D Graphics	Interactive visualization of the elevator environment
	TypeScript	Programming Language	Structured development with type safety
	MQTT Protocol	Communication	IoT data transmission
	Tailwind CSS	UI Styling	Rapid interface development
	ECharts	Data Visualization	Dynamic dashboards and charts
	Vite	Development Environment	Modern build tooling

- **Three.js**

  A powerful WebGL-based 3D graphics library used to build the interactive visualization of the elevator environment. It supports camera controls, mesh animations, material effects, and dynamic scene manipulation.

- **TypeScript**

  Provides static type checking and structured development, helping maintain code quality in a complex application involving multiple real-time data sources and UI components.

- **MQTT Protocol**

  A lightweight and efficient publish-subscribe messaging protocol well-suited for IoT communication. It ensures minimal latency and reliable data delivery between sensors, the broker, and the frontend.

- **Tailwind CSS & ECharts**

  Tailwind CSS is used for utility-first UI styling, allowing rapid interface prototyping and consistency across components. ECharts supports dynamic dashboards and real-time data visualizations such as sensor line charts and heatmaps.

- **Vite Development Environment**

  The application was built and served using **Vite**, a modern frontend build tool that offers fast startup, lightning-fast hot module replacement (HMR), and optimized output. Vite simplifies integration with TypeScript, PostCSS, and TailwindCSS, improving the overall developer

experience.

This combination of tools ensures that the Digital Twin platform is scalable, responsive, and maintainable — capable of handling both static visualization and dynamic real-time interactions.

3.6 Conclusion

This chapter presented the approach taken to architect the Digital Twin platform for LMEs. The system is based on a Unified Namespace architecture, which supports real-time communication among sensors, the 3D visualization interface, and cloud analytics.

There was a two-pronged strategy towards data integration: initially, mock data was utilized to validate the platform, and afterwards real-time sensor integration was achieved through MQTT. Three.js, TypeScript, MQTT, Tailwind CSS, and Vite were utilized to realize the layered architecture of the platform—acquisition, integration, and visualization.

This systematic approach enabled an extensible basis for real-time monitoring and predictive maintenance capabilities in the future. The following chapter outlines the implementation of the prototype and the findings of applying this methodology in practice.

Chapter 4. Prototype development

4.1 Introduction

In this section, we describe the development of the Digital Twin platform for monitoring a Linear Motor elevator Prototype. We conducted the development in three main steps. First, the LME model was designed and modeled using Autodesk Fusion 360 and Blender. In the second phase we develop the Digital Twin platform. We simulated the sensors by creating a utility that generates mock data. Finally, the third stage was linking the Digital Twins to the Unified Namespace (UNS) for real-time sensor data streams.

4.2 Design of the 3D model of Linear Motor elevator Prototype

The development of the 3D model is an important phase of the Digital Twin. The 3D model is the core particularity of the platform. It serves as a reference for visualization of the sensors' data.
The LME prototype model is designed to accurately represent the actual system.
The model should:

- Represent the exact geometric form and function of each part
- Represent the exact physical and mechanical properties of the parts for future physics integration
- Be modular for future interaction

4.2.1 Component of the LME Prototype

The LME Prototype is a small-scale prototype of Linear Motor elevator of 2500 mm height mainly compose of three main parts: the movers, the stators, the guide rails and structure (see Figure 12). The model is designed in the software Autodesk Fusion 360.

4.2.1.1 The movers

The movers are important parts of the Linear Motor elevator; they are the parts that support the elevator cabin and move along the stators to serve floors (see Figure 10). The mover is composed of two fingers that have a set of 8 magnet blocks in each finger. The current LME prototype has two movers.

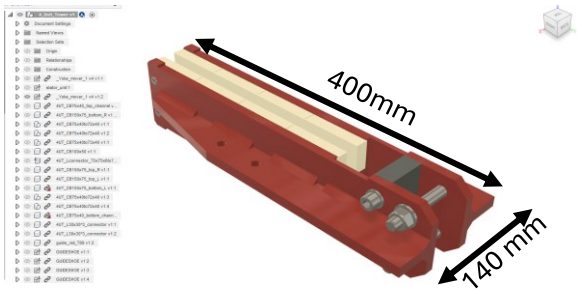


Figure 10: Mover

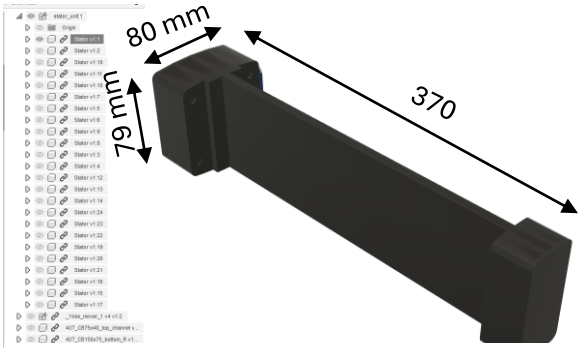


Figure 11: Stator

4.2.1.2 The stators

The stators are the static part of the Linear Motor elevator (see Figure 11). It is a set of winded coils following a specific pattern. The prototype is composed of 24 stators.

4.2.1.3 The guide rails and structure

The guide rails are composed of two profiles that support the vertical movement and alignment of the movers via two guide shoes. The structure consists of multiple channel profiles with various section designed to support the stators and ensure the stability of the whole component during driving. All the components were modeled according to design standards.

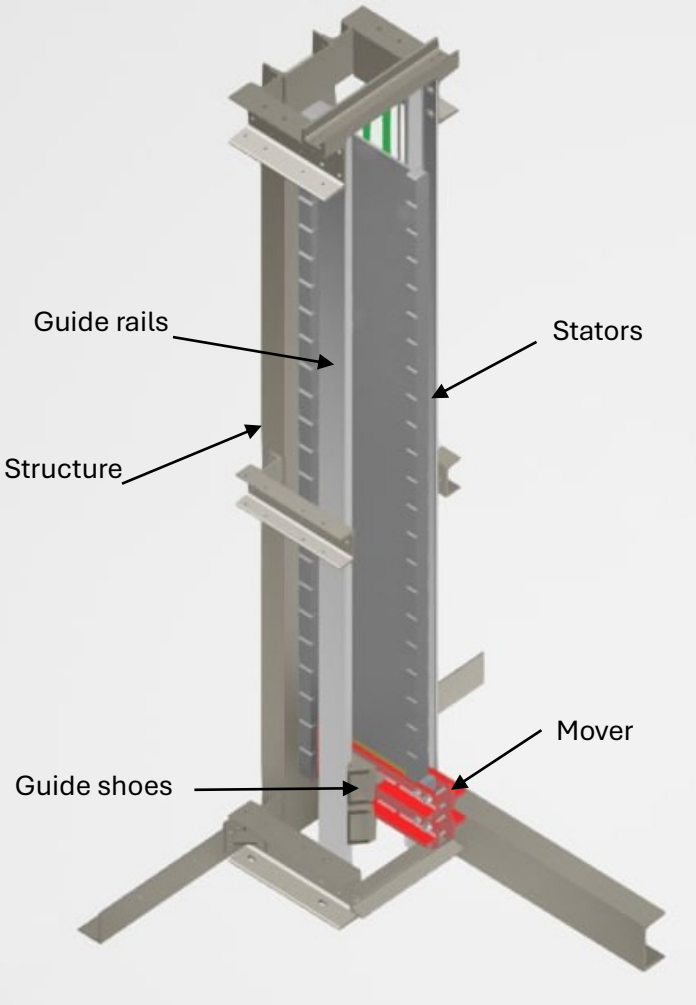


Figure 12: Linear Motor elevator Prototype 3D model

4.2.1.4 Exporting Workflow

To be supported in Three.js, the model design in Fusion 360 should be properly converted to an adequate format.

Export from Fusion 360 to STEP

The LME model was first exported as an assembly in .STEP format. This step ensured that the geometry and assembly configuration would remain intact through the export and transfer process.

Convert STEP to GLTF

The .STEP file was then exported to .GLTF using a free online 3D model converter [https://convert3d.org/convert/step]. This enabled the generation of content for use with Three.js.

Mesh renaming and cleanup in Blender

The GLTF model was imported into Blender, where the meshes were renamed for consistent labeling and clarity (see Figure 13). This step was critical for the ability to perform data binding and real-time interaction within the Digital Twin. Some of the naming conventions used were:

Stator_1 to Stator_24, Mover_1, Mover_2

GuideRail_Left and GuideRail_Right

GuideShoe_Top and GuideShoe_Bottom

This workflow downstream would ensure each component was readily identifiable during runtime, should be updated or monitored separately, and colored based on live sensor data. The final output was a GLTF model which was lightweight, well-structured, and ready for deployment with Three.js

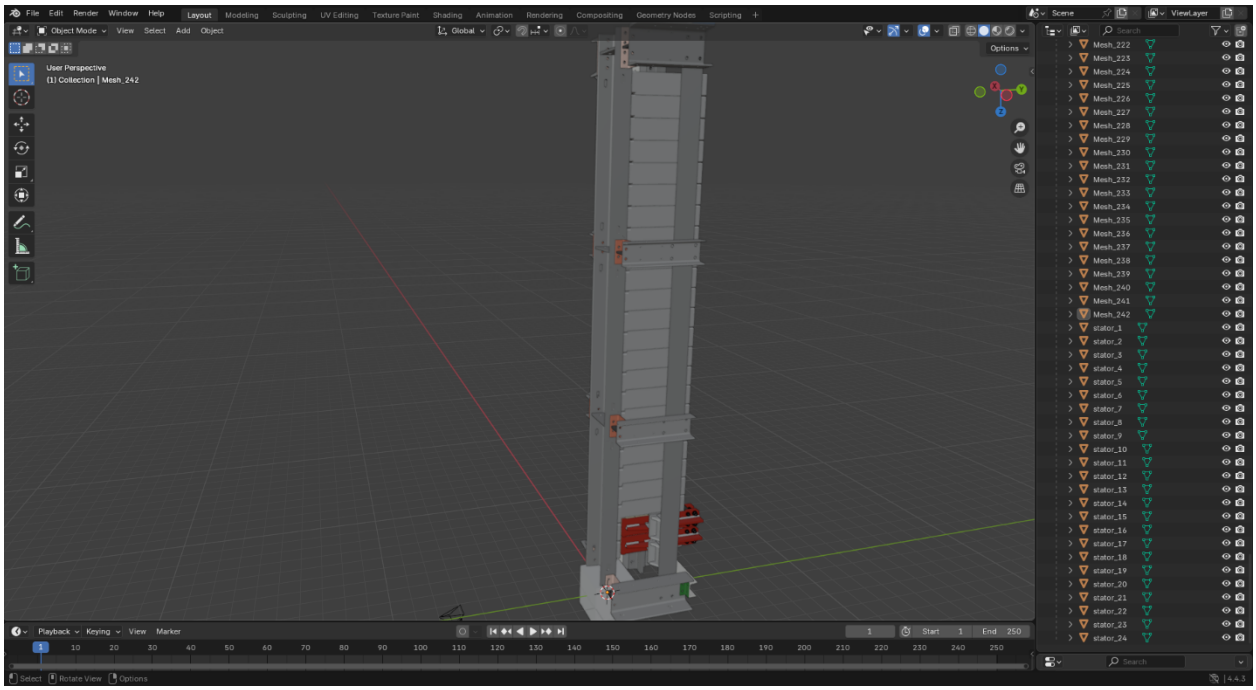


Figure 13: Blender interface - Renaming and grouping mesh

4.3 Digital Twin platform Development

4.3.1 Project Requirements and Goals

Our Digital Twin platform for LMEs aims to provide predictive maintenance and real-time performance monitoring. This section states the initial requirements and objectives for the project and then uses them as a basis for both development and future implementations.

The platform includes mock data integration in the initial approach (refer to 3.4.1), allowing for development and testing outside of a hardware dependent process; however, it is expected this method will allow a clear transition to go from theoretical to practical use of Three.js and TypeScript.

4.3.1.1 Functional requirement

The functional requirements define the end user's basics needs from the software. We have defined here the requirements for the core components of our system.

4.3.1.1.1 3D model visualization

The system provides a 3D model visualization component that delivers an immersive and interactive representation of the Linear Motor elevator system. This visualization enables users to explore elevator components, understand spatial relationships, and monitor status in real-time. Table 3 defines the required features and implementation approaches for the 3D model visualization.

Model Loading and Rendering: The platform shall efficiently load and render the Linear Motor elevator GLTF model with support for complex model hierarchies. It must use the a .glb model file format for optimal balance between detail and performance. Component identification shall be implemented through consistent naming patterns to enable targeted interactions and data association.

Camera and Navigation Controls: The platform shall provide intuitive navigation controls including orbit, pan, and zoom functionality. The implementation must support smooth camera transitions when components are selected. Cameras shall be configurable with an adjustable field of view and near/far planes to accommodate different viewing preferences.

Object Selection and Highlighting: The system shall enable interactive selection of model components through raycasting. Selected components must be visually distinguished using BoxHelperWrap with different visual states for hover and selection. The implementation shall provide an intuitive interaction model for identifying and focusing on specific areas of interest.

Table 3: Functional requirements - 3D model

Feature	Description	Implementation Details
Model Loading and Rendering	• Load and render the Linear Motor elevator GLTF model • Support for complex model hierarchies • Optimized rendering for performance	• Uses .glb model file • Component identification via naming patterns
Camera and Navigation Controls	• Orbit controls for model exploration • Smooth camera transitions • Configurable camera parameters	• Orbit, pan, zoom functionality • Focus transitions on component selection • Adjustable field of view, near/far planes
Object Selection and Highlighting	• Interactive selection of model components • Visual highlighting • Multiple selection states	• Raycasting for selection • BoxHelperWrap for highlighting • Hover and selected states

4.3.1.1.2 Sensor Simulation System

The Digital Twins platform include a Sensor Simulation System that provides the data foundation for monitoring and analyzing the Linear Motor elevator's performance. This component generate, manage, and visualize sensor data that mirrors the behavior of physical sensors in a real elevator installation. Table 4 defines the required features and implementation approaches for the Sensor Simulation System.

Mock Data Generation: The platform shall provide mock data generation capabilities to enable platform functionality during development and testing phases. The sensor type system must be extensible to allow the addition of new sensor types such as vibration or acoustic sensors etc...

Time-Series Data Management: The system shall store and provide access to historical sensor data for trend analysis and pattern recognition. The implementation must use a sliding window approach with configurable history length to maintain recent sensor readings while preventing memory overflow.

Real-Time Updates: The Digital Twins platform shall provide users with current information about the elevator system's status to enable timely decision-making. The implementation must update sensor data at regular intervals (default 1000ms) to create a continuous stream of information

Table 4: Functional requirements - Sensor simulation

Feature	Description	Implementation Details
Mock Data Generation	• Temperature sensor simulation • Configurable data patterns • Support for multiple sensor types	• 20-30°C temperature range • Anomaly injection capability • Extensible sensor type system

Time-Series Data Management	• Historical data storage • Sliding window approach • Fixed-size data arrays	• Configurable history length • FIFO data management • MAX_DATA_POINTS limit (20)
Real-Time Updates	• Periodic data updates • Event-based notification • Synchronized visualization	• setInterval (1000ms) • EventBus for notifications • Coordinated UI updates

4.3.1.1.3 Interactive elements

The platform provides Interactive elements that transform static visualization into a dynamic and engaging user experience. These elements allow users to interact with the 3D model and sensor data in intuitive ways to extract meaningful insights and focus on areas of interest. Table 5 defines the required features and implementation approaches for the interactive elements.

Sensor Visualization: The system shall provide clear visual indicators of sensor locations and status within the 3D model. The implementation must use sprite-based indicators positioned at stator locations to ensure visibility from any viewing angle. The system shall implement visual differentiation through color and shape coding to allow users to quickly identify sensor types and states, with red indicating abnormal conditions. The implementation must include dynamic visual feedback, such as scale and opacity changes, to highlight sensors experiencing significant changes or anomalies.

Hover and Selection Effects: The system shall implement an intuitive interaction model to help users navigate the 3D environment. The implementation must provide immediate visual feedback when users hover over components to indicate interactivity. The system shall include click selection mechanisms with automatic highlighting to clearly distinguish selected items. Camera transitions must automatically adjust to focus on selected components, ensuring users can easily examine areas of interest in detail.

Information Panels: The system shall complement the 3D visualization with detailed textual and graphical information about selected components and sensors. The implementation must include dynamic UI panels that display real-time data alongside historical context. The system shall present component-specific information in a clear and organized manner, including sensors readings, component specifications, and maintenance history. This combination of visualization and information panels must create a comprehensive monitoring environment that supports informed decision-making.

Table 5: Functional requirements - Interactive elements

Feature	Description	Implementation Details
Sensor Visualization	• Sprite-based indicators • Visual differentiation • Dynamic visual feedback	• Positioned at stator locations • Color/shape coding by type and state • Scale and opacity changes
Hover and Selection Effects	• Hover effects • Selection mechanism • Camera transitions	• Visual feedback on hover • Click selection with highlighting • Automatic camera focus
Information Panels	• Dynamic UI panels • Real-time data display	• Detailed sensor information • Historical data context

	• Component-specific information	• Relevant component details

Figure 14 defines the overall system workflow.

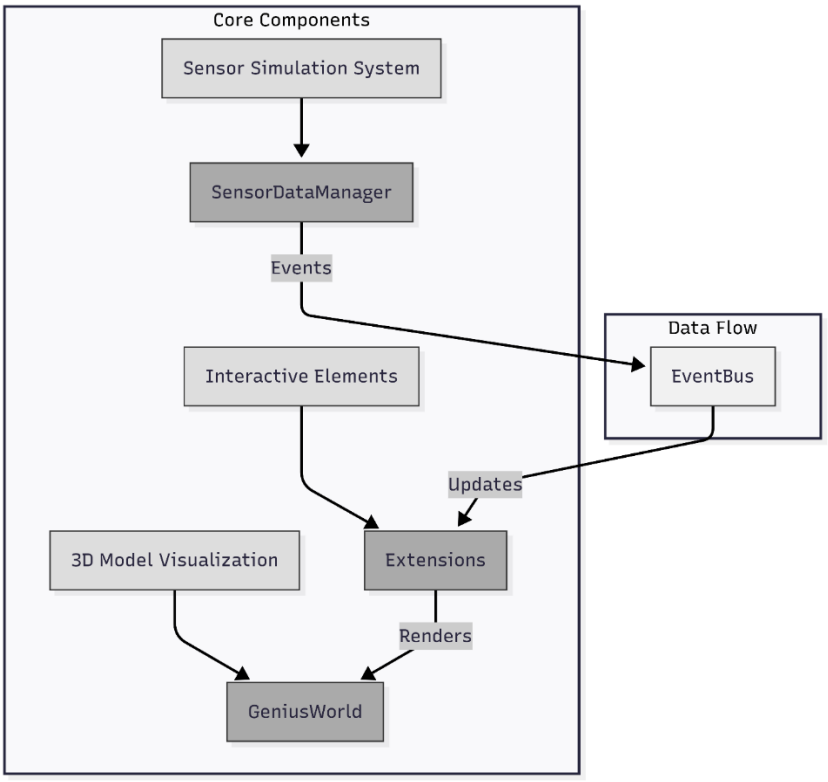


Figure 14: Overall system workflow

4.3.1.2 Non-functional requirements

The non-functional requirements on the other hand are quality constraints that the system must satisfy. We define three main requirements: modular architecture, scalable and future proof data structure and performance optimization.

4.3.1.2.1 Modular architecture

The platform has a modular architecture that divides the system into distinct, interchangeable components that can be developed, tested, and maintained independently. Table 6 defines the required attributes and implementation approaches for the modular architecture.

Extension System: The system must provide an extension system that allows new functionality to be added without modifying the core system. The implementation must also support a standardized lifecycle with well-defined initialization, activation, and disposal phases to ensure consistent behavior.

Component Separation: The platform should implement careful decoupling of system elements, with each component having clear responsibilities and well-defined interfaces. The implementation must separate the GeniusWorld core (providing essential 3D rendering capabilities) from business logic in dedicated modules.

Event-Driven Communication: The system must utilize event-driven communication to allow components to interact without direct dependencies. The implementation must also include a centralized event bus using Mitt to provide type-safe event handling

Table 6: Non-functional requirements - Modular architecture

Attribute	Requirements	Implementation Approach
Extension System	• Pluggable functionality • Standardized lifecycle • Separation of concerns	• Plugin-based architecture • Lifecycle methods (init, update, activate, deactivate, dispose) • Clear separation of rendering and business logic
Component Separation	• Decoupled components • Well-defined interfaces • Testable design	• GeniusWorld logic separation • Interface-based communication • Dependency injection patterns
Event-Driven Communication	• Centralized event handling • Type-safe events • Consistent event structure	• Mitt event bus implementation • TypeScript type definitions for events • Standardized event naming and payload formats

4.3.1.2.2 Performance optimization

The system implements performance optimization techniques to deliver a responsive and immersive user experience. We focus on three key areas: rendering efficiency, interaction responsiveness, and resource management (see Table 7).

Rendering Efficiency: The system must provide efficient rendering for complex elevator models, even on less powerful devices. The implementation should also utilize Three.js optimization techniques components to reduce memory usage and rendering overhead.

Interaction Responsiveness: The platform should ensure users experience fluid and immediate feedback with the actual system.

Resource Management: The system should implement careful memory management and proper cleanup procedures.

Table 7: Non-functional requirements - Performance optimization

Attribute	Requirements	Implementation Approach
Rendering Efficiency	• Optimized scene management • Efficient object rendering • Level-of-detail handling	• Three.js optimization techniques • Object instancing for repeated elements • Progressive loading for complex models
Interaction Responsiveness	• Fast object selection • Smooth interaction handling • Fluid animations	• Efficient raycasting algorithms • Debounced event handling • Optimized transition animations
Resource Management	• Efficient memory usage • Proper resource cleanup • Optimized asset handling	• Memory-efficient data structures • Proper disposal of Three.js objects • Asset loading and caching strategies

4.3.1.2.3 Future proof and scalability

The system implements future-proofing and scalability features to ensure adaptation to evolving requirements and growing datasets. The platform should be a system that remains viable and effective as technology and business needs change (see Table 8).

Standardized Data Formats: The system shall implement standardized data formats to provide a foundation

for future extensibility.

Scalability Considerations: The system shall support growth in both data volume and system complexity. The implementation must include a sensor management architecture capable of accommodating hundreds of sensors without performance degradation, using efficient data structures optimized for large datasets. Real Data Integration Readiness: The platform must ensure a smooth transition from simulated to real-world data sources. The implementation must maintain consistent internal data structures regardless of whether data comes from simulations or physical sensors.

Table 8: Non-fiunctional requirements - Future proof

Attribute	Requirements	Implementation Approach
Standardized Data Formats	• Consistent sensor data • Well-defined metadata • Extensible data models	• HistoricalDataView interface • Structured sensor and component metadata • Flexible type system for future sensors
Scalability Considerations	• Support for growth • Efficient data handling • Effective data presentation	• Scalable sensor management • Optimized data structures for large datasets • Pagination and filtering capabilities
Real Data Integration Readiness	• Seamless transition path • Abstracted data sources • Flexible data adapters	• Consistent internal data structures • Source-agnostic data layer • Configurable format adapters

Figure 15 is a summary of the non-functional requirements

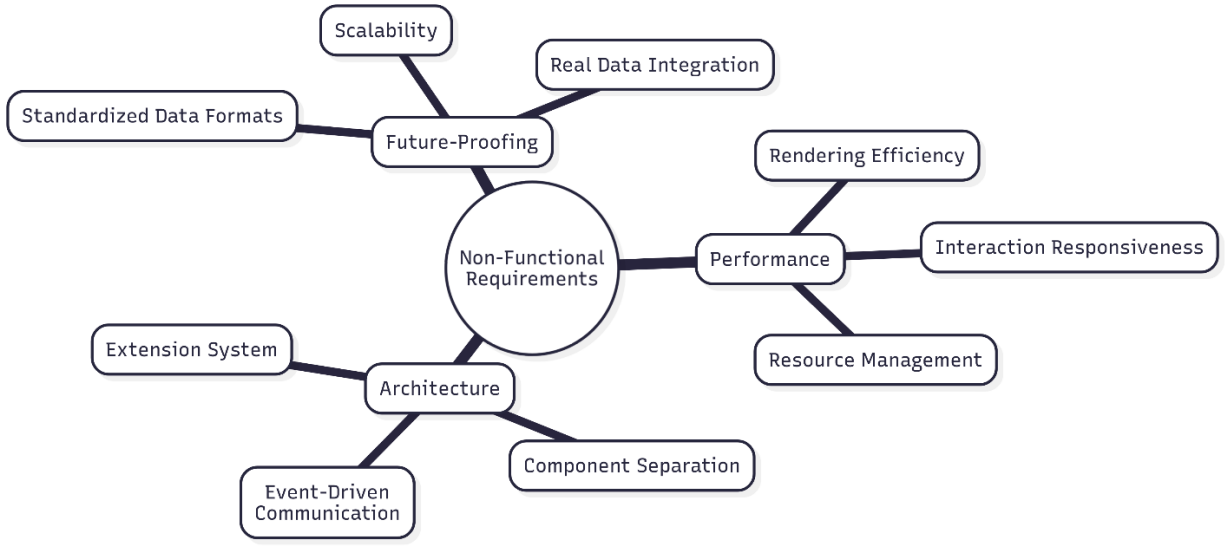


Figure 15: Non-functional requirements summary

4.3.1.3 Project objectives

The Genius-Digital Twins project has been structured with a progressive roadmap of objectives spanning short, medium, and long-term horizons. These objectives are designed to guide the development process, establish clear milestones, and ensure that the platform evolves to meet both immediate needs and future aspirations. By categorizing objectives into these time frames, we create a strategic framework that balances

immediate deliverables with long-term vision

Each objective has been carefully considered to build upon previous achievements, gradually expanding the platform's capabilities from a functional prototype to a full-featured predictive maintenance system. This incremental approach allows for continuous validation and refinement of the platform while delivering value at each stage of development.

4.3.1.3.1 Short-Term Objectives

- **Initiate an operational Digital Twin prototype:** Create a working version utilizing the model to simulate core Digital Twins functions such as visualization and other interactions.
- **Create simulation of mock data (temperature sensor):** We can use simulated sensor data to challenge the model's data pipeline and visual responses without having any physical devices.
- **Create an interactive 3D visualization experience:** Expose the elevator system in the user's web browser using Three.js and interface to explore and interact with each component.
- **Create a modular extendable code base:** Structure the code to accommodate future expansion (additional sensors, predictive models with minimal to no modifications to the existing code).
- **Show value of Digital Twins for elevator maintenance:** Confirm that the Digital Twin technology brings value in supporting visibility, diagnosis, and maintenance planning of elevator systems.

4.3.1.3.2 Mid-Term objectives

- **Integrate real sensor data from LME systems:** Transition from mock data to live streams coming from physical sensors to allow real-time monitoring.
- **Enable predictive analytics for fault prediction:** Use historical data and real-time data to predict faults and recommend preventative actions.
- **Support more types of sensors (e.g., vibration, acoustic):** Add the capability to ingest other types of sensors' data to obtain richer monitoring.
- **Define user roles and access permissions:** Create user management to facilitate multiple users and roles (e.g., technicians, supervision, researchers).
- **Be production-grade ready:** Ensure the system is ready for deployment with a focus on performance, stability and scalability, to be used in live elevator systems.

4.3.1.3.3 Long-Term objectives

- **Create a complete predictive maintenance platform:** Bring Genius-Digital Twins to fruition so it can be a complete solution supporting end-to-end industrial maintenance operations
- **Achieve a 50% reduction in elevator downtime:** Use predictive maintenance to decrease unplanned breakdowns and service interruptions significantly.
- **Increase useful life of equipment via smart scheduling:** Help operators plan better interventions that achieve less wear and increase the useful life of the component.
- **Automated tracking for compliance reporting:** Deliver tools that can automatically capture data and create reports for compliance with standards in safety and regulation.
- **Dramatically reduce lifecycle maintenance spend:** Enable better decisions that are derived from data, improve maintenance resource use and avoid catastrophic failures.

4.3.2 System Design and UML Diagrams

In this section we explain the structure and design of the components of the Digital Twin platform using software engineering architectures. The way we employed the design articulates how its characteristics would accommodate the objectives of scalability, modularity, and future-proof.

We used an event-driven architecture to allow components to be coupled, leading to the possibility of extending functionality and easier maintenance. The diagrams provided in this section describe how the components will work together to achieve the designed goals.

4.3.2.1 Use Case Diagram

The Use Case Diagram illustrates how the user interacts with the Digital Twins platform. The actors are Maintenance Engineers and System Administrators, who interact with the system in different ways according to their roles.

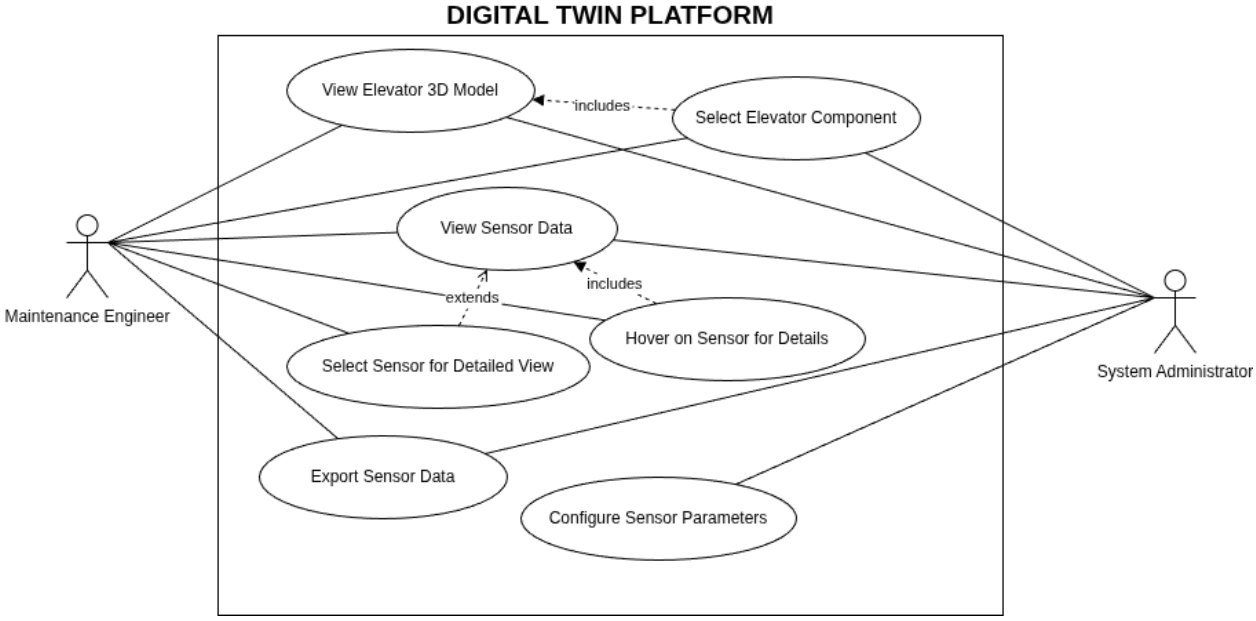


Figure 16: Use case diagram

The diagram (Figure 16) shows that Maintenance Engineers and System Administrators have access to the 3D model and sensor data; however, only System Administrators can configure the sensor parameters. The "extends" and "includes" relationships help visualize how some use cases build on top of others to create a hierarchy of interactions.

4.3.2.2 Components diagram

The Component Diagram resumes the software components/modules of the Digital Twins platform system and demonstrates how they communicate. The system is divided into core components, extensions, and UI components.

Key components include:

GeniusWorld: The central component responsible for 3D scene rendering and interaction

SensorDataManager: Produces and manages simulated sensor data

StatorManager: Manages stator components in the 3D model

MoverManager: Manages mover components in the 3D model

EventBus: Handles communication between modules using an event-driven approach

Extensions: Various extensions that add specific functionality to the system

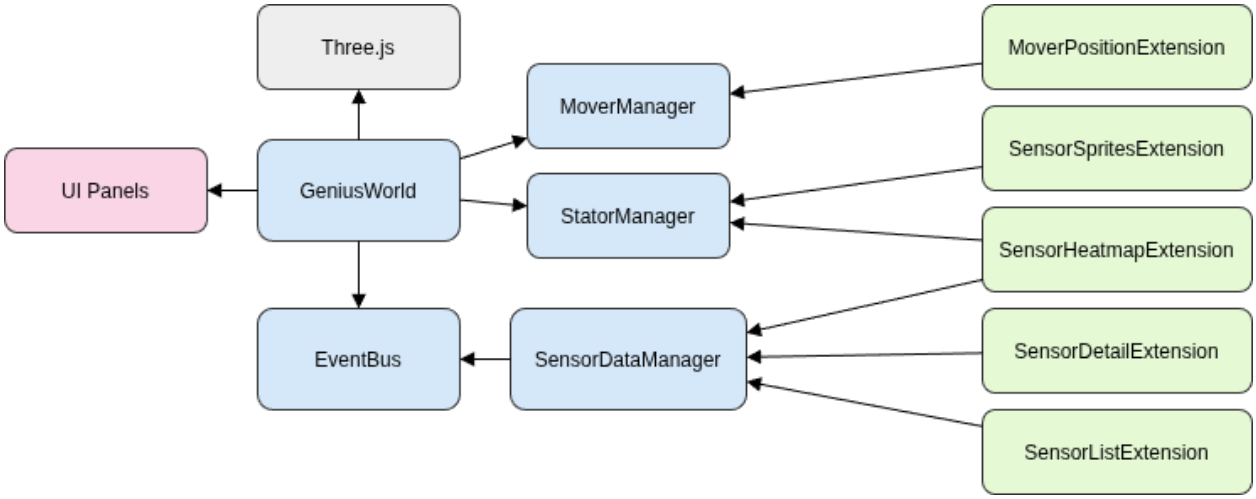


Figure 17: Components diagram

Figure 17 shows how GeniusWorld is the core object that connects to Three.js for 3D rendering and manages multiple extensions. The EventBus allows the objects to communicate with each other, providing a smoother architecture that is loosely coupled. The extensions add specific functionality to the system, including sensor visualization, sensor details, and heatmaps.

4.3.2.3 Entity Relation Diagram

This document presents the Entity Relationship Diagram (ERD) for the Digital Twins platform, illustrating the data model and relationships between classes. The platform uses a data model centered around sensors, channels, and historical data samples. This ERD provides a visual representation of these entities and their relationships, helping them to understand how data is structured and flows through the system.

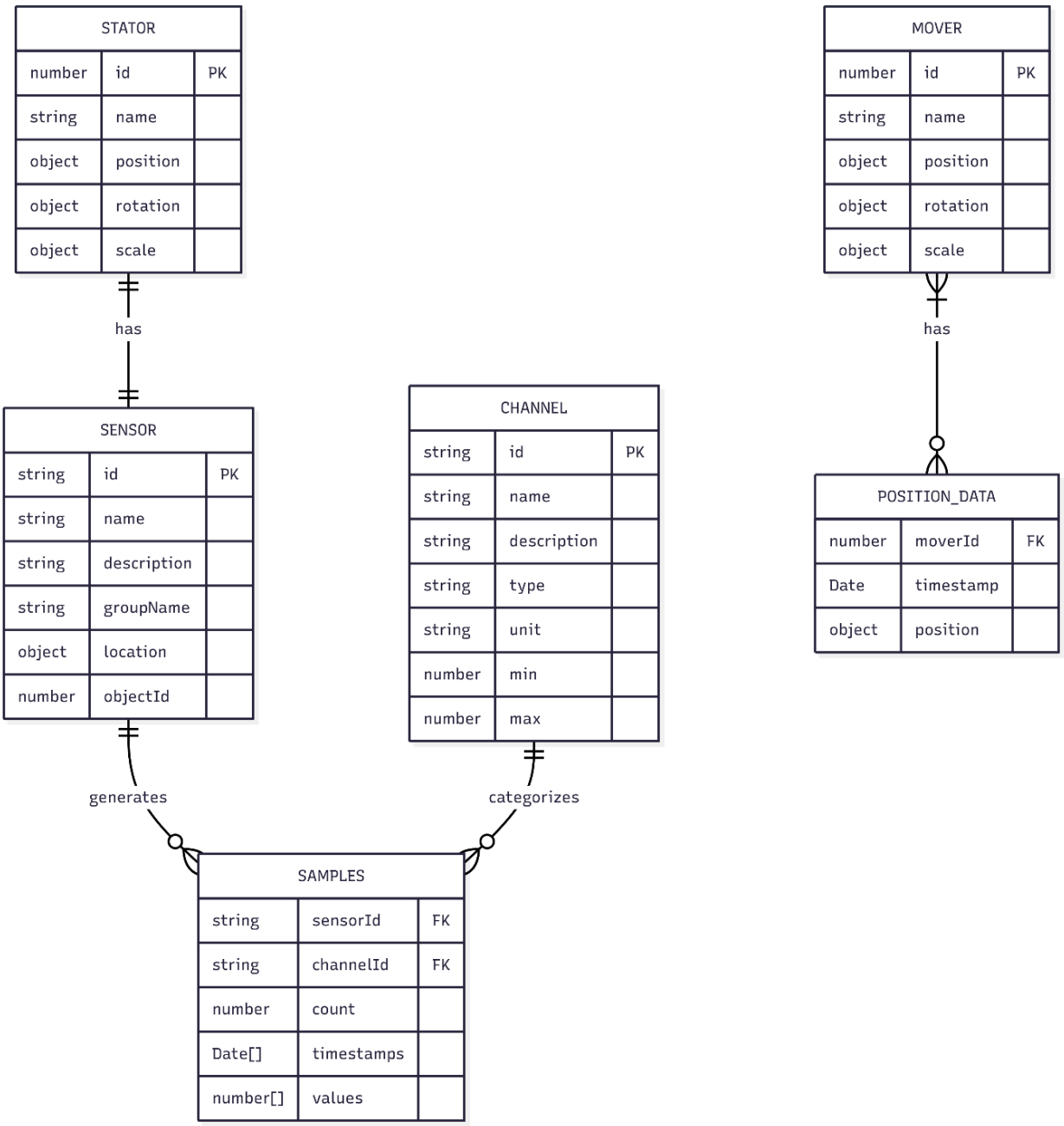


Figure 18: Entity Relation diagram

4.3.2.3.1 Entity Descriptions

Sensor

Represents an IoT sensor in the system. Each sensor is associated with a specific stator component in the 3D model and can generate data for multiple channels (e.g., temperature, vibration).

Channel

Represents a type of measurement that sensors can provide. Each channel has a specific data type, unit, and valid range (min/max values).

Samples

Represents historical data collected from sensors. Each sample is associated with a specific sensor and

channel and contains a series of timestamped values.

Stator

Represents a stator component in the elevator's 3D model. Each stator has one associated sensor.

Mover

Represents a mover component in the elevator's 3D model. Movers can change position over time.

Position Data

Represents historical position data for movers, tracking how they move over time.

4.3.2.3.2 Relationship Descriptions

Sensor to Samples: One-to-many relationship. A sensor can generate multiple samples, one for each channel it supports.

Channel to Samples: One-to-many relationship. A channel can have multiple samples, one for each sensor that supports it.

Stator to Sensor: One-to-one relationship. Each stator has exactly one associated sensor.

Mover to Position Data: One-to-many relationship. A mover can have multiple position data points over time.

4.3.2.3.3 Implementation Notes

In the actual implementation:

Sensors are stored in a *Map<SensorID, Sensor>*

Channels are stored in a *Map<ChannelID, Channel>*

Samples are stored in a nested Map structure: *Map<SensorID, Map<ChannelID, Samples>>*

The *SensorDataManager* class implements the *HistoricalDataView* interface to provide access to this data. This structure allows for efficient lookup of sensor data by sensor ID and channel ID, which is important for real-time visualization and analysis.

4.3.2.4 Sequence Diagram

The Sequence Diagram illustrates the data flow lifecycle in the Genius-Digital Twins system, focusing on how mock sensor data is generated, distributed, and visualized (see Figure 19). It also shows how user interactions trigger events and updates in the system.

The data flow can be summarized as follows:

- SensorDataManager generates mock sensor data at regular intervals
- EventBus broadcasts update events to interesting components
- SensorSpritesExtension and SensorDetailExtension process and visualize the data
- User interactions (hover, click) trigger additional events and updates

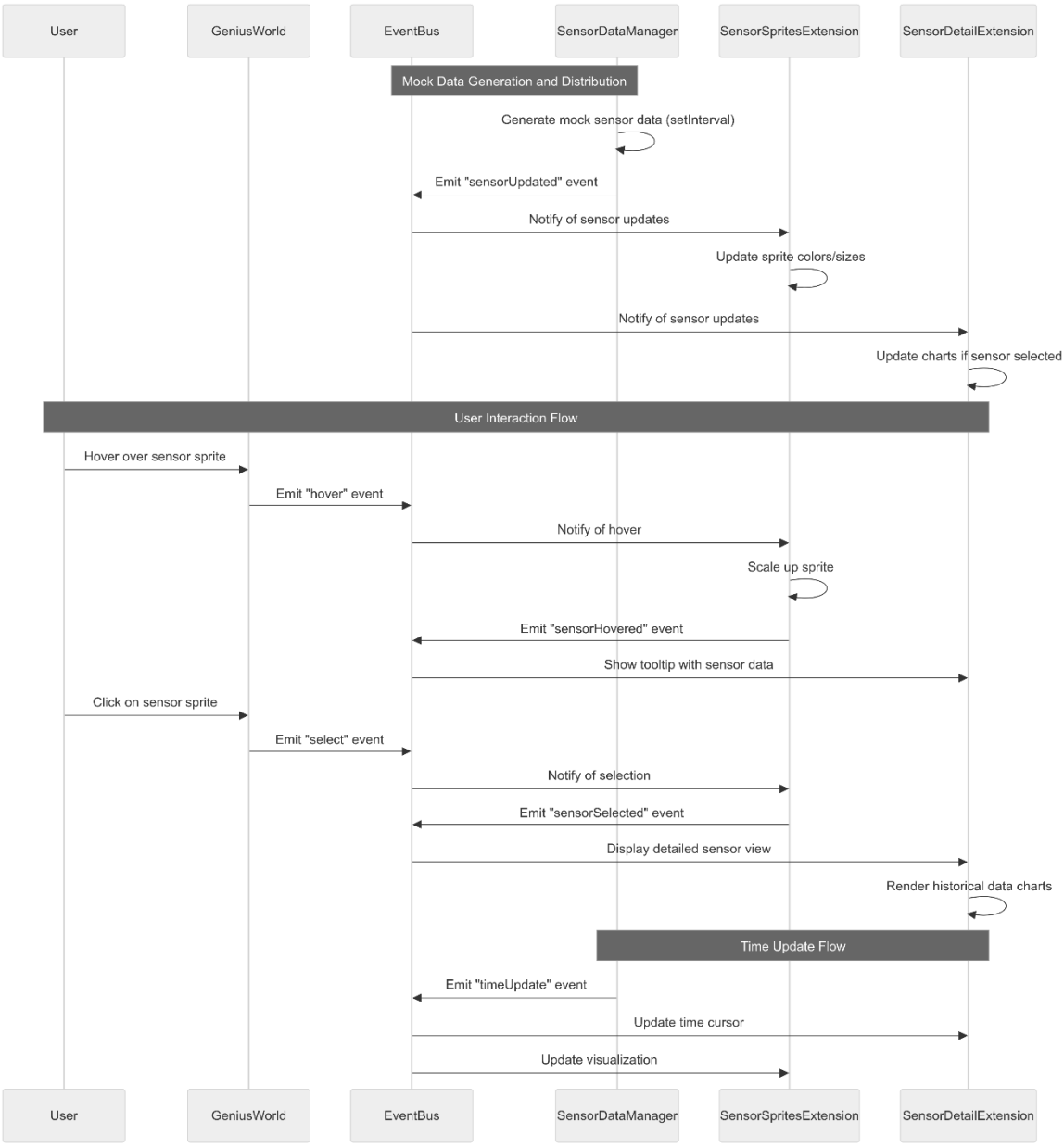


Figure 19: Sequence diagram

The sequence diagram clearly shows the event-driven nature of the system, with events flowing through the EventBus to coordinate updates between components. It also illustrates how user interactions trigger a chain of events that result in visual updates to the interface.

4.3.3 Development implementation

This section outlines the construction and deployment of the Digital Twin (Digital Twins) platform, illustrated through the development of a Digital Twins prototype for LMEs using mock data. Subsequently, a modular and adaptable system architecture is implemented, with a strong emphasis on real-time visualization and interactive capabilities.

All code excerpts referenced in the following sections are provided in the appendices for detailed examination.

4.3.3.1 Development Environment Setup

4.3.3.1.1 Vite and TypeScript Configuration

The project is built with Vite Framework. Its provide fast development server startup and hot module replacement. The configuration in vite.config.ts sets up path aliases and TailwindCSS integration. We configured TypeScript with strict type checking and modern ECMAScript features, to ensure type safety throughout the application.

4.3.3.1.2 Three.js Integration

The core 3D rendering is carried out in the GeniusWorld class, which encapsulates Three.js initialization. The rendering loop is run with requestAnimationFrame to ensure smooth animations and updates.

4.3.3.1.3 3D model Integration

We load 3D models using Three.js's GLTFLoader, with error handling, progress tracking, and automatic registration of objects to Raycaster:

4.3.3.1.4 Object Identification and Classification

The platform identifies and classifies objects in the 3D model based on naming patterns. For example, the SensorSpritesExtension identifies stators to place sensor sprites:

4.3.3.1.5 Mock Data Simulation System

The SensorDataManager class generates mock data for temperature sensors using setInterval to periodically send updates.

4.3.3.1.6 Sensor Types and Parameters

The system defines sensor types and parameters with configurable ranges and units.

4.3.3.1.7 Event-Driven Architecture

The platform uses the mitt library to implement a centralized event bus for communication between decoupled components. The components communicate through events for all interactions and updates.

4.3.3.1.8 Extension System

Base Extension Class: We use an extension system to add functionality in a modular way. The SensorExtension class serves as the base class for all sensor extensions:

Specialized Extensions: Specialized extensions extend the base class to provide specific functionality. For example, the SensorSpritesExtension visualizes sensors as sprites in the 3D scene:

Heatmap Implementation: We implemented a heatmap visualization for temperature data using the SensorHeatmapExtension class, which colors stators based on temperature readings:

4.3.3.1.9 Mover Implementation

The platform includes mover animation using the MoverPositionExtension class, which provides smooth transitions between floors:

The extension also includes audio feedback when the mover reaches its destination:

4.3.3.1.10 Object Wrapper Implementation

The platform uses an object wrapper (Object3DWrap) to extend Three.js Object3D functionality with animation and movement tracking:

The wrapper uses tweens for smooth animations:

4.3.3.1.11 Raycaster Implementation

The platform implements interactive object selection and hovering using the GeniusRaycaster class:

The raycaster handles both mesh and sprite objects, applying different visual effects for each:

4.3.3.1.12 User Interface Components

Sensor Visualization Techniques: The platform uses sprite-based visualization for sensors, with dynamic scaling and opacity to create interactive feedback.

Interactive Elements: The platform implements interactive elements such as hover and selection effects, with camera transitions to focus on selected components:

Control Panels and Dashboards: The platform implements UI panels for displaying sensor information, with real-time updates based on data changes:

The development and implementation of the Digital Twins platform demonstrates a modular, extensible architecture with a strong focus on real-time visualization and interaction. The use of modern web technologies including Three.js, TypeScript, and Vite enables efficient development and a responsive user experience. The event-driven architecture and extension system provides flexibility for future enhancements and integration with real sensor data.

The Digital Twin Platform represents the core technological infrastructure of our system, serving as the bridge between physical elevator components and their virtual representations. It is a sophisticated software framework that integrates multiple technologies to create an interactive, real-time digital replica of the Linear Motor elevator system. This platform is designed with a layered architecture that facilitates data flow, visualization, and analysis.

At its heart, the Digital Twin Platform consists of three primary components: the 3D Visualization Interface built with Three.js, which renders the elevator model with high fidelity; the BIM 3D model Integration, which ensures accurate representation of the physical components; and the Dashboard & Web Viewer, which provides an intuitive user interface for monitoring and interaction. These components work in concert to deliver a comprehensive view of the elevator system's status and performance.

The platform subscribes to data from the Unified Namespace (MQTT Broker) in the Integration Layer, enabling real-time updates from both equipment-level and environmental sensors. This bidirectional communication allows not only for visualization of current states but also for user interactions that can influence the physical system. The platform's modular design ensures scalability and extensibility, allowing for future enhancements and integration with additional technologies as the system evolves.

Figure 20 is a screenshot of the Digital Twin platform

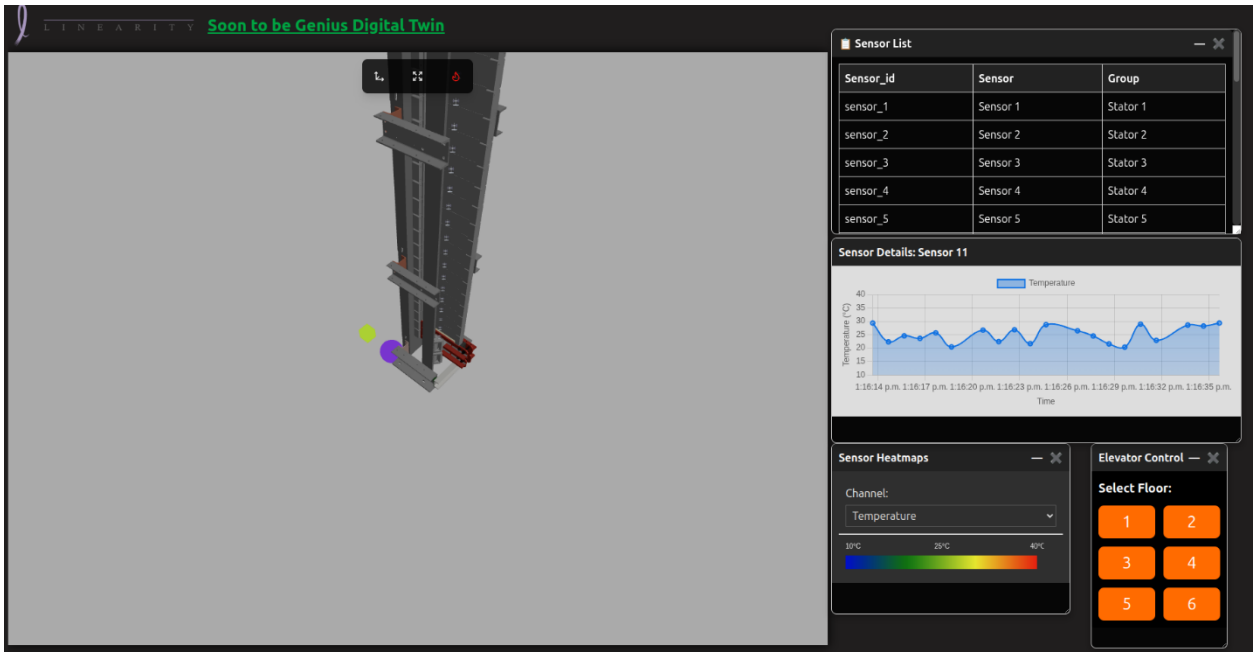


Figure 20: Digital Twin Platform

4.4 Integration with UNS

4.4.1 Introduction

The transition from mock data to live sensor integration represents a critical evolution in the Genius-Digital Twins Digital Twin platform. While mock data provides a controlled environment for development and testing, real-time sensor data enables the Digital Twin to accurately reflect the current state of the physical system.

The Unified Namespace (UNS) serves as the communication backbone of our system, providing a standardized way to organize and access data from various sensors and systems. By leveraging MQTT (Message Queuing Telemetry Transport) as our communication protocol, we create a lightweight, publish-subscribe network that efficiently handles real-time data streams from multiple sensors.

The primary objective of this integration is to enable live sensor streams to dynamically update the 3D visualization, creating a responsive Digital Twin that accurately represents the current state of the physical system. This real-time capability is essential for monitoring, diagnostics, and predictive maintenance applications.

4.4.2 MQTT Broker and Unified Namespace Setup

4.4.2.1 MQTT Broker Selection and Configuration

For our implementation, we've selected Mosquitto as our MQTT broker due to its lightweight nature, reliability, and open-source availability. However, the system is designed to be compatible with any MQTT broker.

Basic configuration parameters include:

Port: 1883 (standard MQTT) and 9001 (MQTT over WebSockets)

Authentication: Username/password authentication

QoS Settings: Quality of Service level 1 (at least once delivery)

Persistence: Enabled for message durability

Retain Messages: Enabled for last known values

4.4.2.2 Unified Namespace Structure

The Unified Namespace represents a structured topic tree that organizes all data channels in a hierarchical, discoverable format. This approach provides several benefits:

Discoverability: New sensors and data channels can be easily discovered

Organization: Logical grouping of related data

Filtering: Efficient subscription to specific data subsets

Scalability: Easy addition of new sensors and data types

Figure 21 shows the general structure followed by the UNS:

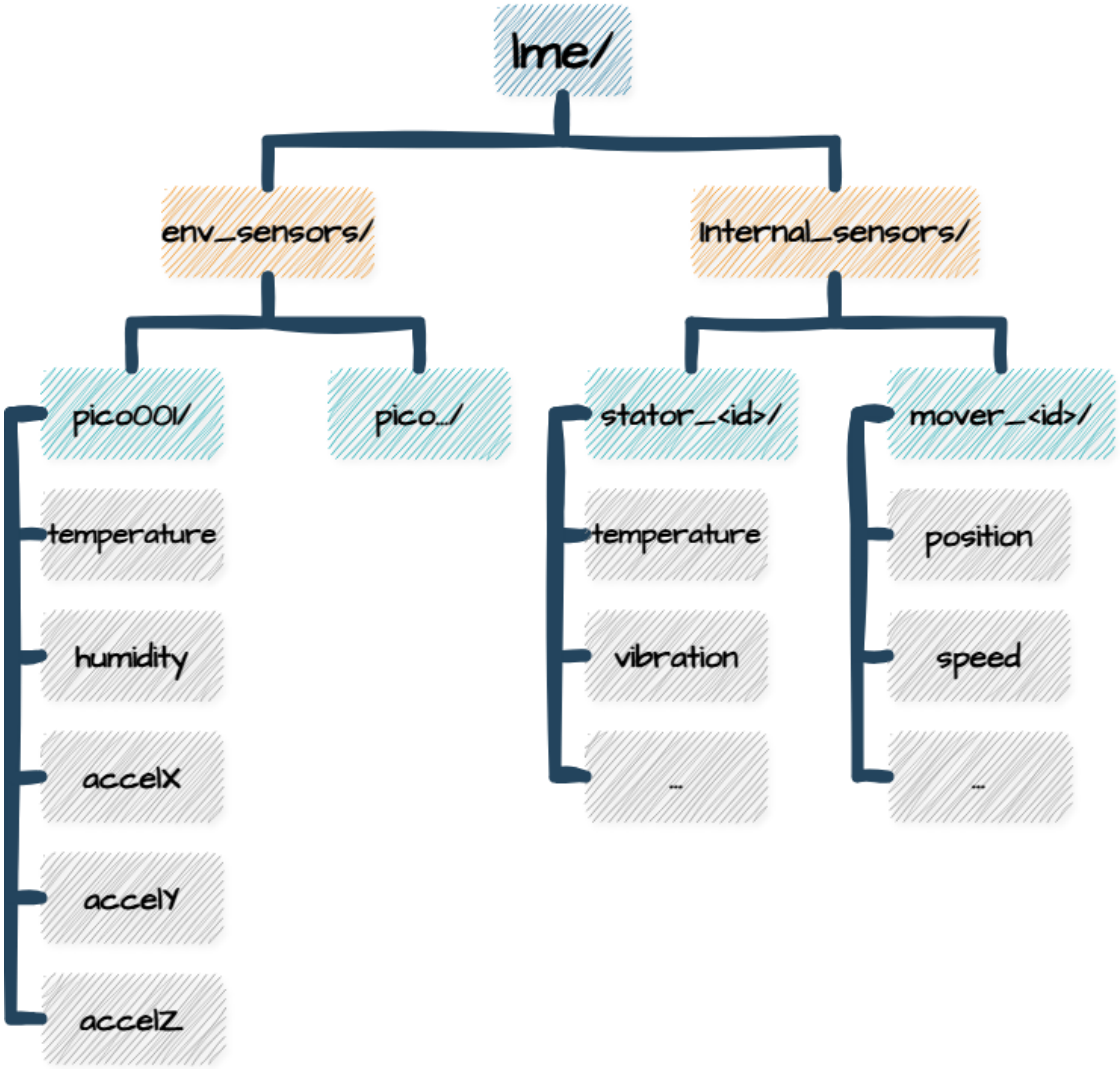


Figure 21: UNS Structure

This structure allows clients to subscribe to specific subsets of data using wildcards. For example:
lme/internal_sensors/stator_+/temperature subscribes to temperature data from all stators
lme/env_sensors/pico001/# subscribes to all data from the pico001 environmental sensor

4.4.3 Replacing Mock Data with Real-Time Streams

4.4.3.1 Strategy for Transition

Our strategy for transitioning from mock data to real-time streams focuses on simplicity and reliability:

- Direct Integration: Implementing a straightforward integration of MQTT data into the SensorDataManager
- Complete Replacement: Fully replacing mock data generation with real-time sensor data
- Consistent Data Structure: Maintaining the same internal data structures for both development and production
- Seamless Visualization: Ensuring visualization components work identically with real data as they did with mock data

Implementation in SensorDataManager

The SensorDataManager has been enhanced to support both mock data generation and MQTT subscription:

```
1. class SensorDataManager implements HistoricalDataView {
2.     private sensors: Map<SensorID, Sensor> = new Map();
3.     private channels: Map<ChannelID, Channel> = new Map();
4.     private sensorSamples: Map<SensorID, Map<ChannelID, Samples>> = new Map();
5.     constructor() {
6.         this.init();
7.     }
8.     private init() {
9.         this._initializeSensors();
10.        this._initializeChannels();
11.        // Connect to MQTT broker
12.        mqttClient.connect();
13.        // Set up MQTT message handler
14.        eventBus.on('mqttMessage', this.handleMQTTMessage.bind(this));
15.    }
16.    // Handle incoming MQTT messages
17.    private handleMQTTMessage({ topic, payload }: { topic: string, payload: any }): void {
18.        // Extract sensor ID and channel ID from topic
19.        const topicParts = topic.split('/');
20.        const sensorType = topicParts[2]; // e.g., 'stator', 'environment'
21.        const sensorId = topicParts[3];   // e.g., '1', 'pico001'
22.        const channelId = topicParts[4];  // e.g., 'temperature', 'humidity'
```

```
23.         // Map external sensor ID to internal sensor ID
24.         let internalSensorId: SensorID;
25.         if (sensorType === 'stator') {
26.             internalSensorId = `sensor_${sensorId}`;
27.         } else {
28.             internalSensorId = `${sensorType}_${sensorId}`;
29.         }
30.         // Update sensor data
31.         this.updateSensorData(internalSensorId, channelId, payload.value, new
Date(payload.timestamp));
32.     }
33. }
```

4.4.3.2 Mapping MQTT Topics to Internal Data Structures

To integrate MQTT data with our existing data structures, we map MQTT topics to sensor IDs and channel IDs:

Topic: lme/sensors/stator_1/temperature

Sensor ID: sensor_1 (mapped from stator_1)

Channel ID: temperature

Topic: lme/sensors/environment/pico001/humidity

Sensor ID: environment_pico001

Channel ID: humidity

This mapping allows the application to maintain its existing data structure while incorporating real-time data from MQTT.

4.4.4 Topic Naming Conventions and Data Payload Format

4.4.4.1 Hierarchical Topic Structure

Our topic naming convention follows a hierarchical structure aligned with UNS standards:

lme/sensors/<sensor_type>_<id>/<channel>

Examples:

lme/internal_sensors/stator_3/temperature

lme/env_sensors/ pico001/humidity

lme/internal_sensors/mover_2/position

This structure provides clear organization and enables efficient filtering using MQTT wildcards.

4.4.4.2 JSON Payload Format

All sensor data is transmitted using a consistent JSON payload format:

{

"sensorId": "stator_3",
"channel": "temperature",
"value": 27.8,
"timestamp": "2025-06-27T12:31:00Z",
"unit": "celsius",
"sensor_type": "DHT11"
}

Key fields include:

sensorId: Unique identifier for the sensor

channelId: The data channel (e.g., temperature, humidity)

value: The measured value

timestamp: ISO 8601 formatted timestamp

unit: The unit of measurement

sensor_type: The type of sensor (optional)

Additional fields may be included for specific sensor types or to provide metadata.

4.4.5 Real Sensor Testing and Integration

4.4.5.1 Hardware Setup

Our testing environment includes real sensors connected to microcontrollers that publish data to the MQTT broker:

ESP32 with MQTT Client Library:

MicroPython implementation

Connects to WiFi and MQTT broker

Publishes sensor data at regular intervals

Sensors:

DHT11: Temperature and humidity sensor

MPU6050: 3-axis accelerometer for vibration monitoring

Additional sensors can be added as needed

4.4.5.2 Sensor Calibration and MQTT Publishing

The sensor code includes calibration and publishing logic:

```
1. def read_and_publish_sensors():
2.     # Read DHT11 sensor
3.     dht_sensor.measure()
4.     temp = dht_sensor.temperature()
5.     hum = dht_sensor.humidity()
6.     # Read MPU6050 sensor
7.     x, y, z = mpu.read_accel_data()
```

```
    # Prepare and publish temperature data
    temp_data = {
        "timestamp": iso_time,
        "value": temp,
        "unit": "celsius",
        "sensor_type": "DHT11",
        "sensor_id": "pico001-temp"
    }
    # Publish to MQTT
    client.publish(f"{BASE_TOPIC}/temperature", ujson.dumps(temp_data))
```

Example Topics and Messages

During testing, the following topics and messages are published:

Topic: lme/sensors/environment/pico001/temperature

{
"timestamp": "2025-06-27T12:31:00Z",
"value": 24.5,
"unit": "celsius",
"sensor_type": "DHT11",
"sensor_id": "pico001-temp"
}

Topic: lme/sensors/environment/pico001/humidity

{
"timestamp": "2025-06-27T12:31:00Z",
"value": 45.0,
"unit": "percent",
"sensor_type": "DHT11",
"sensor_id": "pico001-hum"
}

Topic: lme/sensors/environment/pico001/accel/x

{
"timestamp": "2025-06-27T12:31:00Z",
"value": 0.02,
"unit": "g",
"sensor_type": "MPU6050",
"sensor_id": "pico001-accel",
"axis": "x"
}

Chapter 5. Verification and results

To verify the Digital Twin platform, we conducted research both in developing countries and developed countries. As part of our research, we conducted research with elevator installation and maintenance companies in Benin. We have also conducted testing and verification activities with Linearity, a Linear Motor elevator developing company in Japan.

5.1 Verification in Benin

We work with two major companies, the first called CFAO Infrastructure and is officially representing OTIS in Benin. While the second, PRO elevators is a sub-regional company mainly working with MP elevators, a Spanish elevators manufacturing company.

We conducted a meeting to explain and discuss our research with each company. As result they allow us to conduct data collection on their installations. The data was collected from two buildings and 3 elevators. It consists of, temperature and humidity, vibration and sound data. We intend to use this data as samples for training maintenance prediction AI models.

Figure 22: Data collection building in Benin. Left: MP elevator Motor; Right: Sunu assurance building

5.1.1 Data collection tools

To collect the data, we use three tools:

- The Okudake sensor logger: It is a set of sensors that can collect temperature, humidity, vibration, light and magnetic field. For our purposes, we only use the vibration and temperature and humidity sensor.

Figure 23 shows the sensor placed on top of the motor.

Figure 23: Okudake sensor on MP elevator motor

- A set of sensors is built by us: We have also built a sensor set that collects temperature and humidity and vibration data to collect the data. It consists of an ESP32 microcontroller running with micropython. The temperature and humidity sensor is DHT11 and the vibration sensor is an MPU5060.
  Figure 24 shows the system placed on top of the motor.

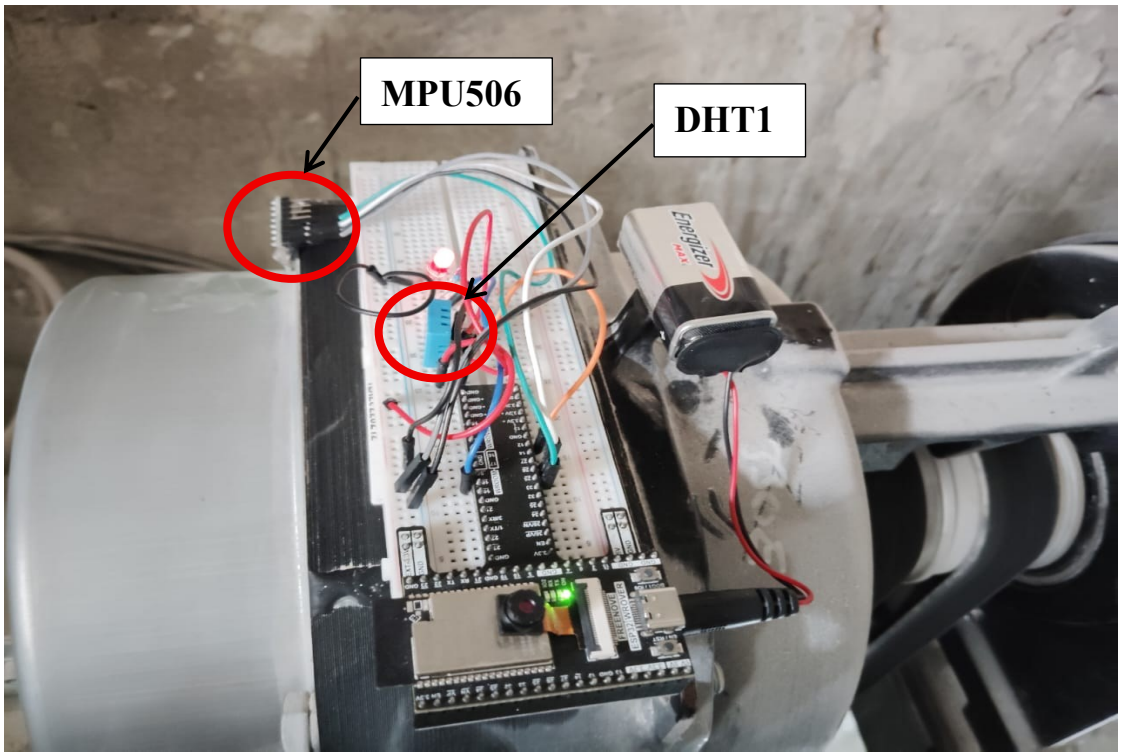


Figure 24: Built sensor set on OTIS elevator Motor in Benin

- An IC Recorder: The recorder is used to record the sound of the motor during driving.
  The recordings will be used to identify normal patterns when the motor is driving and not driving. Allowing us to train AI models that can perform anomaly detection.
  Figure 25 shows the recorder placed on the cabin rooftop.

Figure 25: IC Recorder on OTIS elevator cabin in Benin

5.1.2 Data collection results

The data we collected at the sites are being preprocessed for further usage in training and testing of our AI models.

Figure 27 and Figure 28 present some of the data collected.

Vibration in the 3 axis. The data was recorded during two days from 2PM to 4AM.

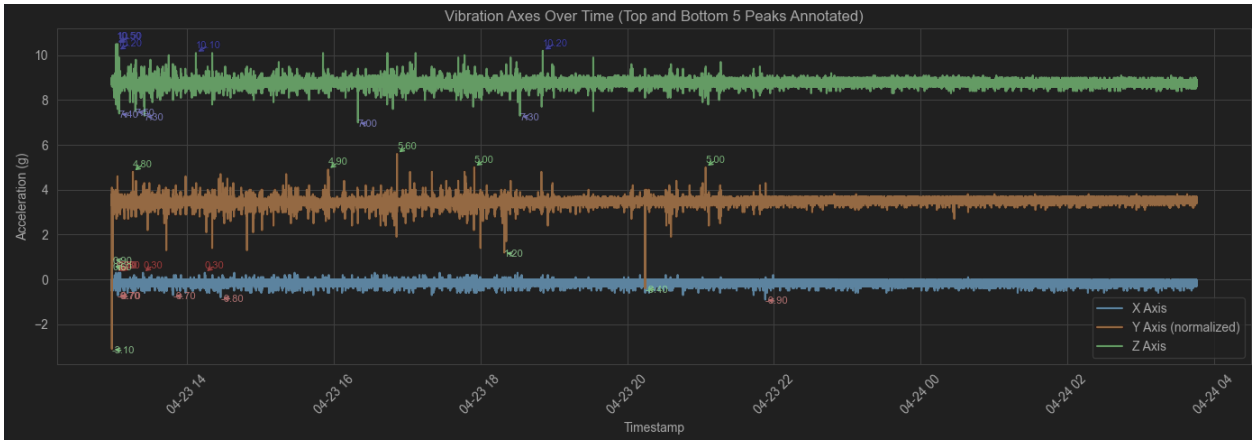


Figure 26: Vibration data on X, Y, Z

Temperature and humidity. The data was recorded during two days from 2PM to 4AM.

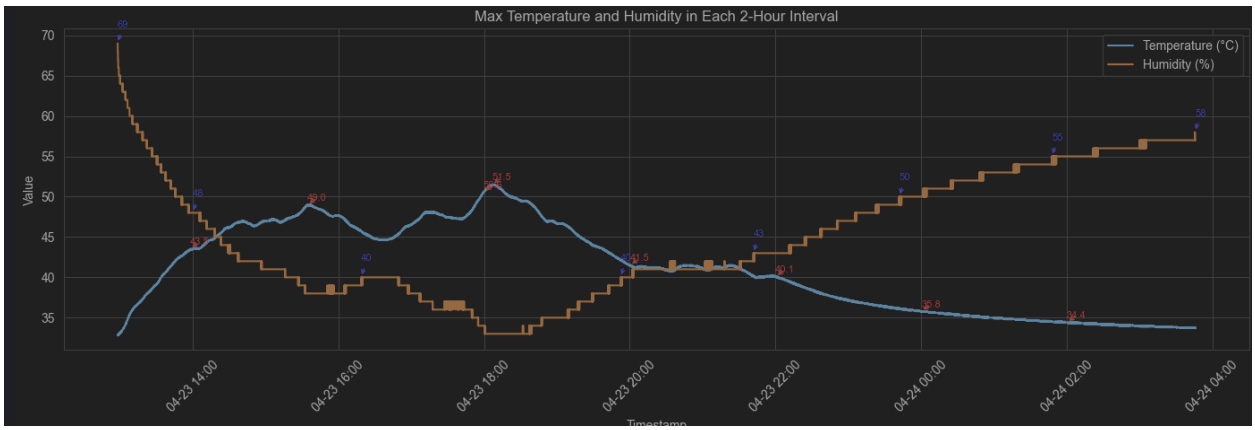


Figure 27: Temperature and humidity data

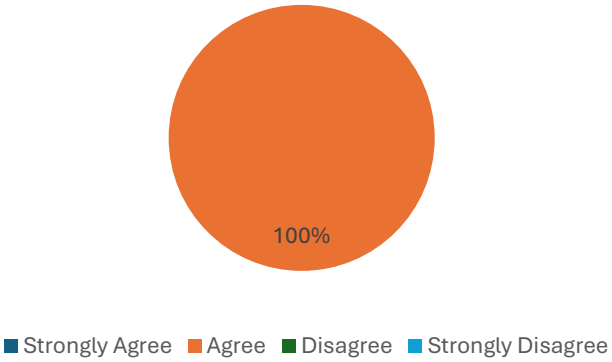
5.1.3 Verification questionnaire

We prepared a simple questionnaire of 5 questions to gather thoughts of elevators professionals in Benin about integrating the Digital Twin in their process.

The questionnaire has 5 questions with multiple choices:

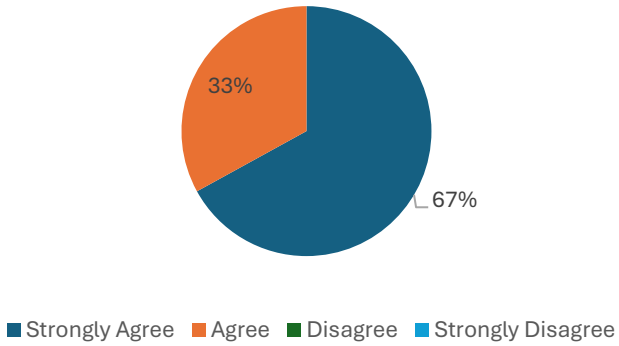
1. **A real-time 3D visualization of elevator components would help me detect potential issues earlier.**

Option	*Responses*	*Percentage*
Strongly Agree	0	0%
Agree	3	100%
Disagree	0	0%
Strongly Disagree	0	0%



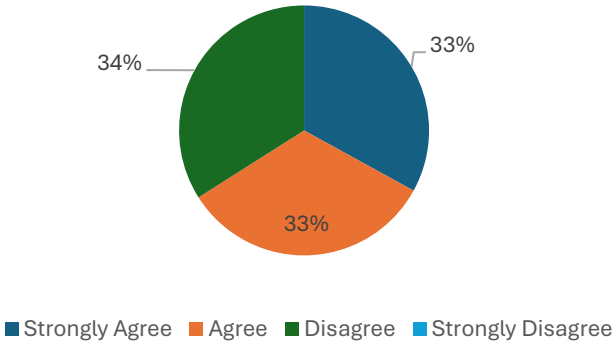
2. **Integrating more sensor data (temperature, vibration, current, voltage, sound, etc.) into a Digital Twin would improve our ability to perform predictive maintenance.**

Option	*Responses*	*Percentage*
Strongly Agree	2	67%
Agree	1	33%
Disagree	0	0%
Strongly Disagree	0	0%



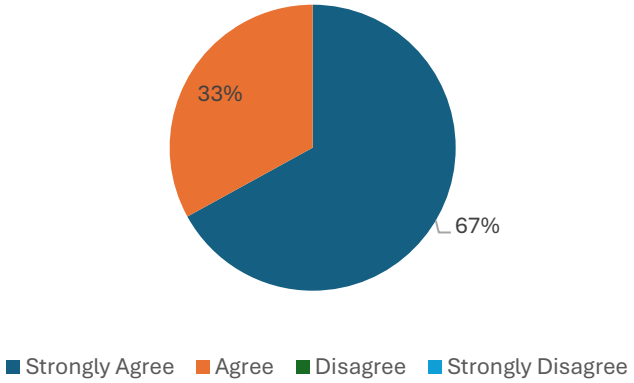
3. **A Digital Twin could reduce the frequency of unexpected elevator downtimes in our operations.**

Option	*Responses*	*Percentage*
Strongly Agree	1	33%
Agree	1	33%
Disagree	1	34%
Strongly Disagree	0	0%



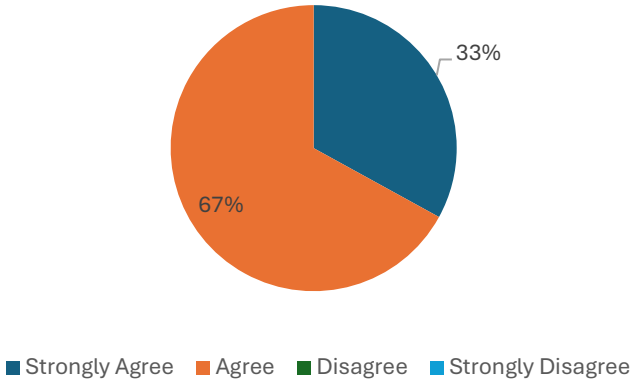
4. **I would use a Digital Twin if it allowed me to simulate elevator failures and test responses without interrupting service.**

Option	*Responses*	*Percentage*
Strongly Agree	2	67%
Agree	1	33%
Disagree	0	0%
Strongly Disagree	0	0%



5. **Having a centralized interface (Digital Twin Dashboard) for monitoring all elevators would make my maintenance tasks more efficient.**

Option	*Responses*	*Percentage*
Strongly Agree	1	33%
Agree	2	67%
Disagree	0	0%
Strongly Disagree	0	0%



5.1.4 Summary of the verification in Benin

The survey results show that elevator maintenance professionals in Benin see real value in using Digital Twin technology. They agreed that a real-time 3D view of elevator components would help spot problems earlier. Also adding more sensor data such as temperature or vibration could make predictive maintenance easier and more accurate. Most respondents said they would use a Digital Twin if it let them test failures without stopping the elevator, and they believe a centralized dashboard would make their work more efficient. Overall, the feedback supports the idea that Digital Twins could really help improve elevator maintenance. Our prototype is then valuable to them, and future improvements also need to be considered.

5.2 Verification at Linearity

A verification has been conducted. We connected directly through the Unified Namespace our Digital Twin platform to Linear Motor elevator prototype at Linearity. The testing also includes an environment sensor setup that monitors temperature humidity and vibration.

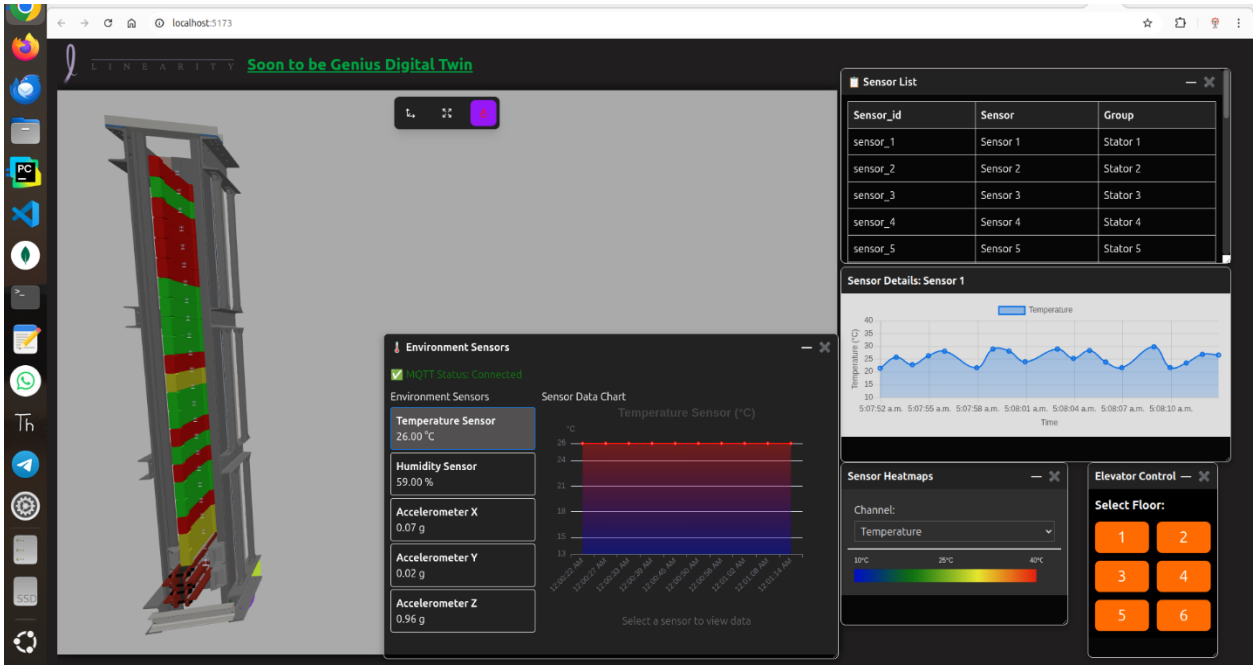


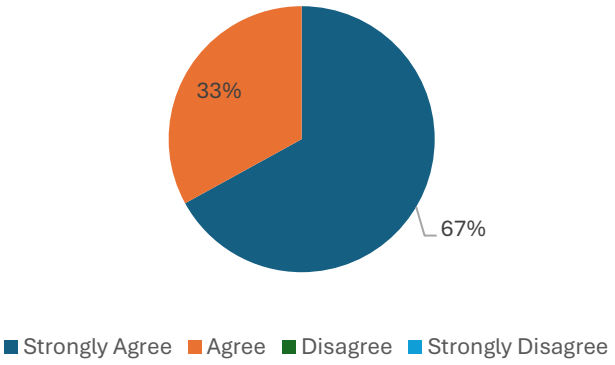
Figure 28: Digital Twin Platform

A questionnaire has also been conducted with the staff at Linearity to enquire about the relevance of our Digital Twin prototype in their workflows.

The questionnaire results are as described.

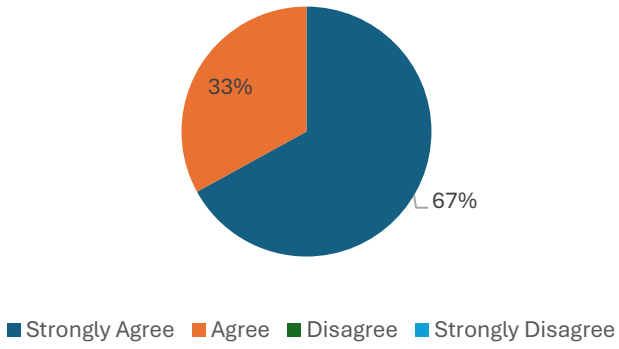
1. **A real-time 3D visualization of elevator components would help me detect potential issues earlier.**

Option	*Responses*	*Percentage*
Strongly Agree	2	67%
Agree	1	33%
Disagree	0	0%
Strongly Disagree	0	0%



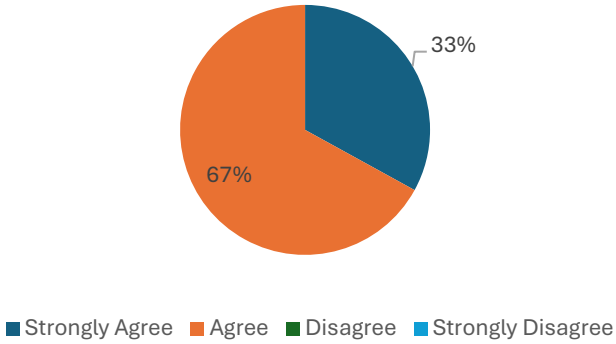
2. **Integrating more sensor data (temperature, vibration, current, voltage, sound, etc.) into a Digital Twin would improve our ability to perform predictive maintenance.**

Option	*Responses*	*Percentage*
Strongly Agree	2	67%
Agree	1	33%
Disagree	0	0%
Strongly Disagree	0	0%



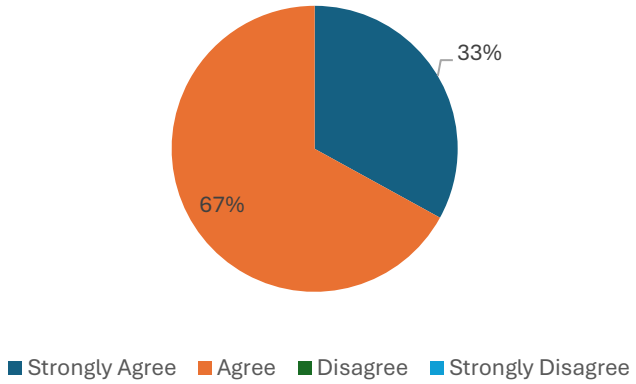
3. **A Digital Twin could reduce the frequency of unexpected elevator downtimes in our operations.**

Option	*Responses*	*Percentage*
Strongly Agree	1	33%
Agree	2	67%
Disagree	0	0%
Strongly Disagree	0	0%



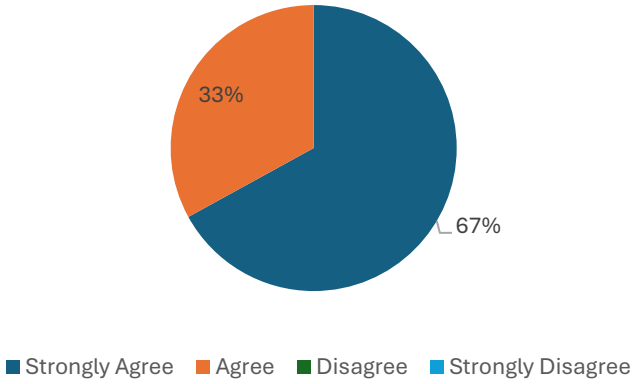
4. **I would use a Digital Twin if it allowed me to simulate elevator failures and test responses without interrupting service.**

Option	*Responses*	*Percentage*
Strongly Agree	1	33%
Agree	2	67%
Disagree	0	0%
Strongly Disagree	0	0%



5. **Having a centralized interface (Digital Twin Dashboard) for monitoring all elevators would make my maintenance tasks more efficient.**

Option	*Responses*	*Percentage*
Strongly Agree	2	67%
Agree	1	33%
Disagree	0	0%
Strongly Disagree	0	0%



Chapter 6. Conclusion

Our study has successfully demonstrated the feasibility of applying Digital Twin technology for building facilities, especially for elevators. By integrating real-time sensor data, and 3D visualization, the developed prototype enables real-time monitoring. The approach has relevance to buildings in rapidly developing urban cities, where traditional approaches to maintenance fall short due to complexity, resource limitations, and operational pressures.

The platform demonstrated promising accuracy and responsiveness. Vibration, temperature, and acoustic data were collected using low-cost sensors and correctly processed using machine learning models, achieving predictive insight independent of premium industrial platforms. The Three.js-based browser-centered 3D interface proved effective at representing both the structural condition and sensor input, creating an intuitive bridge between physical and digital elevator assets.

But field tests also revealed aspects to optimize further. Ambient noise, data latency, and sensor physical placement continue to be important variables for the reliability of forecasting. Additionally, scalability and robustness over long-term use, especially in varying environmental conditions, must be addressed to take it from prototype to real-world deployment.

Nevertheless, this study lays a solid foundation for future development. The incorporation of more precise sensors, the extension of data processing models, and the addition of capabilities to support other building systems, for instance the HVAC or the structural health monitoring—can potentially increase operational efficiency and sustainability even further. With the continued evolution of IoT, AI, and 3D web technologies, Digital Twins have the potential to transform building maintenance strategies, reduce downtime, and increase asset lifespan—particularly in developing regions with visions for intelligent, efficient infrastructure.

Appendix

A. Development Environment Setup

1) Vite and TypeScript Configuration

```
1. // vite.config.ts
2. import { defineConfig } from 'vite'
3. import tailwindcss from '@tailwindcss/vite'
4. import path from 'path'
5. export default defineConfig({
6.   // Use root path for Vercel, or '/genius-dt/' for GitHub Pages
7.   base: process.env.DEPLOY_TARGET === 'gh-pages' ? '/genius-dt/' : '/',
8.   plugins: [
9.     tailwindcss(),
10.   ],
11.   build: {
12.     chunkSizeWarningLimit: 10000, // Increased from default 500
13.   },
14.   resolve: {
15.     alias: {
16.       "@": path.resolve(__dirname, './src')
17.     }
18.   }
19. })
20. ```TypeScript is configured with strict type checking and modern ECMAScript features, ensuring type safety throughout the application:```
21. // tsconfig.json (partial)
22. {
23.   "compilerOptions": {
24.     "target": "ES2020",
25.     "module": "ESNext",
26.     "strict": true,
27.     "noUnusedLocals": true,
28.     "noUnusedParameters": true,
29.     "paths": {
30.       "@/*": ["src/*"]
31.     }
32.   }
33. }
```

2) Three.js Integration

```
1. // GeniusWorld.ts (constructor)
2. constructor(element: HTMLElement) {
3.     this.element = element;
4.     // Renderer
5.     this.renderer = new WebGLRenderer({
6.         antialias: true,
7.         alpha: true,
8.     });
9.     this.renderer.setSize(element.clientWidth, element.clientHeight);
10.    element.appendChild(this.renderer.domElement);
11.    // Scene
12.    this.scene = new Scene();
13.    this.scene.background = new Color("#aaaaaa");
14.    // Camera
15.    this.camera = new PerspectiveCamera(75, element.clientWidth / element.clientHeight, 0.1, 100000);
16.    this.camera.position.z = 5;
17.    // Add light
18.    this.addLights();
19.    // Raycaster
20.    this.raycaster = new GeniusRaycaster(this.camera, this.element);
21.    // OrbitControls
```

```
22.     this.orbitControls = new
OrbitControls(this.camera,
this.renderer.domElement);
23.     this.orbitControls.enableDamping =
true;
24.     this.orbitControls.dampingFactor =
0.05;
25. }
26. //  The rendering loop is implemented with
`requestAnimationFrame` to ensure smooth
animation and               // updates:
27. // GeniusWorld.ts (render method)
28. public render(): void {
29.     const rendering = () => {
30.         requestAnimationFrame(rendering);
31.         this.orbitControls.update();
32.         this.renderer.render(this.scene,
this.camera);
33.     };
34.     rendering();
35. }
```

B. 3D model Integration

1) GLTF Model Loading

```
 2. // GeniusWorld.ts (LoadGLTF method)
 3. public async LoadGLTF(fileURL: string):
Promise<Object3D> {
 4.     if (!fileURL) {
 5.         throw new Error("Invalid file
URL");
 6.     }
 7.     return new Promise((resolve, reject) =>
{
 8.         const gltfLoader = new
GLTFLoader();
 9.         const loadingTimeout =
setTimeout(() => {
10.             reject(new Error('GLTF loading
timed out.'));
11.         }, 10000);
12.         this.emitter.emit("loading-start");
13.         gltfLoader.load(
14.             fileURL,
15.             (gltf) => {
16.
clearTimeout(loadingTimeout);
17.                 this.scene.add(gltf.scene);
18.
this.raycastObjects.push(gltf.scene);
19.
this.raycaster.setObjects(this.raycastObjects);
20.
21.                 // Register objects inside
GeniusWorld
22.                 gltf.scene.traverse((child)
=> {
23.                     if (child instanceof
Object3D) {
24.
child.updateMatrixWorld(true); // Force update
world positions
25.
this.raycastObjects.push(child);
26.                         const wrappedObject
= new Object3DWrap(child);
27.
this.object3DWrapMap.set(child.uuid,
wrappedObject);
28.
this.object3DWrapNameMap.set(child.name,
wrappedObject);
29.                     }
30.                 });
32.
this.raycaster.setObjects(this.raycastObjects);
33.
eventBus.emit("modelLoaded", gltf.scene);
```

```
                this.emitter.emit("loading-complete", gltf.scene);
                resolve(gltf.scene);
            },
            (progress) => {
                this.emitter.emit("loading-progress", progress.loaded / progress.total);
            },
            (error) => {
            clearTimeout(loadingTimeout);
                this.emitter.emit("loading-error", error);
                reject(error);
            }
        );
    });
}
```

2) Object Identification and Classification

```

// SensorSpritesExtension.ts (init method)
async init(): Promise<void> {
    super.init();
    this.dataView = getSensorData();

    // Wait for Stator Positions to be Extracted
    // Ensure StatorManager receives an array
    const statorList = this.world.getObject3DWrapList()
        .filter(item => item.object3D.name.startsWith("stator_"))
        .map(item => item.object3D);

    if (!Array.isArray(statorList) || statorList.length === 0) {
        console.warn("⚠ No stators found in the scene.");
        return;
    }
    const statorPositions = StatorManager.extractStatorPositions(statorList);
    if (statorPositions.length === 0) {
        console.warn("⚠ No stator positions found. Sensor sprites will not be created.");
        return;
    }
    // Add sensor sprites after extracting positions
    this.addSensorSprites(statorPositions);
}

```

C. Mock Data Simulation System

1) Data Generation Strategy

```
// sensorUtils.ts (startDataUpdates method)
private startDataUpdates() {
    if (this.updateInterval) {
        clearInterval(this.updateInterval);
    }
    this.updateInterval = setInterval(() => {
        const now = new Date();
        const MAX_DATA_POINTS = 20;
        let updatedSensors: Record<SensorID, number> = {};

        // Function to Update Sensor Samples
        const updateSamples = (sensorId: SensorID, channelId: ChannelID, newValue: number) => {
            const sensorChannelSamples = this.sensorSamples.get(sensorId)?.get(channelId);
```

```
15.             if (!sensorChannelSamples)
return;
17.  sensorChannelSamples.timestamps.push(now);
18. sensorChannelSamples.values.push(newValue);
19. sensorChannelSamples.count++;
21.             // Maintain only the last 20
data points
22.             if
(sensorChannelSamples.timestamps.length >
MAX_DATA_POINTS) {
23. sensorChannelSamples.timestamps.shift();
24. sensorChannelSamples.values.shift();
25.             }
27.             updatedSensors[sensorId] =
newValue; // Store updated values for UI
28.         };
30.         // Update each stator sensor
temperature
31.         for (let i = 1; i <= 24; i++) {
32.             const temperature = 20 +
Math.random() * 10; // Range: 20-30°C
33.             updateSamples(`sensor_${i}`,
"temperature", temperature);
34.         }
36.         // Emit sensor updates
37.         eventBus.emit("sensorUpdated",
updatedSensors);
38.     }, 1000);
39. }
```

2) **Sensor Types and Parameters**

```
 1. // sensorUtils.ts (_initializeChannels
method)
 2. private _initializeChannels() {
 3.     this.channels.set("temperature", {
 4.         name: "Temperature",
 5.         type: "double",
 6.         unit: "°C",
 7.         min: 10,  // Adjusted to real-world
range
 8.         max: 40,
 9.     });
10. }
```

3) **Historical Data Management**

```
 1. // sensorUtils.ts (updateSamples function)
 2. const updateSamples = (sensorId: SensorID,
channelId: ChannelID, newValue: number) => {
 3.     const sensorChannelSamples =
this.sensorSamples.get(sensorId)?.get(channelId
);
 4.     if (!sensorChannelSamples) return;
 6.  sensorChannelSamples.timestamps.push(now);
 7. sensorChannelSamples.values.push(newValue);
 8.     sensorChannelSamples.count++;
10.     // Maintain only the last 20 data
points
11.     if
(sensorChannelSamples.timestamps.length >
MAX_DATA_POINTS) {
12.   sensorChannelSamples.timestamps.shift();
13.   sensorChannelSamples.values.shift();
14.     }
16.     updatedSensors[sensorId] = newValue; //
Store updated values for UI
17. };
```

D. **Event-Driven Architecture**

1) **EventBus Implementation**

```
 1. // Events.ts
 2. import mitt, {Emitter} from "mitt";
 3. import {Object3D, Vector3} from "three";
 4. import {SensorID, ChannelID} from
"./../Extensions/Sensors/HistoricalDataView";
 6. // Define event types
 7. type Events = {
 8.     select: Object3D[];
 9.     deselect: void;
```

```
    hover: Object3D;
    unhover: Object3D | void;
    timeUpdate: Date;
    sensorSelected: SensorID;
    sensorHovered: SensorID;
    sensorUpdated: Record<SensorID, number>;
    heatmapUpdate: { channel: ChannelID; timestamp: Date };
    modelLoaded: Object3D;
    statorPositionsReady: Vector3[];
};
// Create a shared event emitter
const eventBus: Emitter<Events> = mitt<Events>();
export default eventBus;
```

2) Key Events

```
// GeniusWorld.ts (constructor)
eventBus.on("select", (object: Object3D[]) => console.log("Object selected:", object));
eventBus.on("hover", (object) => console.log("Hovered over:", object.name));
eventBus.on("deselect", () => console.log("Deselected object."));
eventBus.on("unhover", () => console.log("Unhovered all."));

// SensorSpritesExtension.ts (init method)
eventBus.on("hover", (hoveredObject) => this.onSpriteHovered(hoveredObject));
eventBus.on("unhover", (hoveredObject) => this.onSpriteUnhovered(hoveredObject));
eventBus.on("select", (selectedObjects) => {
    if (!selectedObjects[0]) return;
  this.onSpriteClicked(selectedObjects[0]);
});
eventBus.on("sensorSelected", (sensorId: SensorID) => {
    // Find the corresponding sprite and highlight it
});

```

E. Extension System

1) Base Extension Class

```
// SensorExtension.ts
export default class SensorExtension {
    protected world: GeniusWorld;
    protected scene: Scene;
    protected active: boolean = false;
    constructor(world: GeniusWorld) {
        this.world = world;
        this.scene = world.scene;
    }
    // Initialize the sensor extension (Called once)
    init(): void {
        this.world.addExtension(this);
        this.active = true;
    }
    // Update data (Runs every frame if needed)
    update(): void {
        if (!this.active) return;
        // Example: Refresh sensor data
    }
    // Activate (Enables extension if disabled)
    activate(): void {
        this.active = true;
    }

    // Deactivate (Disables without disposing)
    deactivate(): void {
```

```
33.         this.active = false;
35.     }
37.     // Dispose (Removes from GeniusWorld)
38.     dispose(): void {
39.         this.deactivate();
40.         this.world.removeExtension(this);
42.     }
43. }
```

2) Specialized Extensions

```
 1. // SensorSpritesExtension.ts (addSensorSprites method)
 2. private addSensorSprites(statorPositions: Vector3[]): void {
5.     // Load texture
 6.     const texture = new TextureLoader().load(this.spriteTextureUrl);
 8.     statorPositions.forEach((position, index) => {
 9.         const sensorId = `sensor_${index + 1}`;
11.         if (this.sensorSprites.has(sensorId)) return; // Prevent duplicates
13.         // Create sprite material
14.         const material = new SpriteMaterial({map: texture, depthTest: false, transparent: true});
15.         const sprite = new Sprite(material);
16.         sprite.scale.set(0.3, 0.2, 0.3); // Adjust size for visibility
18.         // Place sprite at stator position (Offset slightly upward)
19.   sprite.position.copy(position).add(new Vector3(0.1, 0.5, 0));
21.         sprite.name = sensorId; // Assign sensor ID
23.         this.scene.add(sprite);
24.         this.sensorSprites.set(sensorId, sprite); // Store in map
26.   this.world.raycastObjects.push(sprite); // Add sprite to raycast list
29.     });
31.     // Ensure Raycaster Detects Sprites
32.this.world.raycaster.setObjects(this.world.raycastObjects);
34. }
```

F. Advanced Visualization Components

1) Heatmap Implementation

```
 1. // SensorHeatmapExtension.ts (applyHeatmap method)
 2. public applyHeatmap(): void {
 3.     if (!this.dataView) return;
 4.     if (!this.heatmapEnabled) return; // Prevent applying if disabled
 8.     for (let i = 1; i <= 24; i++) {
 9.         const sensorId: SensorID = `sensor_${i}`;
10.         const statorObject = this.world.getObject3DWrapByFullName(`stator_${i}`);
11.         if (!statorObject) {
12.             console.warn(`⚠ Stator ${i} not found.`);
13.             continue;
14.         }
16.         const samples = this.dataView.getSamples(sensorId, this.selectedChannel);
17.         if (!samples || samples.values.length === 0) {
18.             console.warn(`⚠ No data for ${sensorId}.`);
19.             continue;
20.         }
```

```
22.         const latestValue =
samples.values[samples.values.length - 1]; //
Get latest sensor value
23.         const heatmapColor =
this.getColorFromValue(latestValue);
25.   statorObject.object3D.traverse((child:
THREE.Object3D) => {
26.             if (child instanceof THREE.Mesh
&& child.material instanceof
THREE.MeshStandardMaterial) {
27.      const material = child.material as
THREE.MeshStandardMaterial;
28. material.color.set(heatmapColor); //
Directly set color
29.                 material.needsUpdate =
true;
30.             }
31.         });
32.     }
35. }
37. //The heatmap uses a color scale to
represent temperature values, with blue for
cold, green for moderate,  // yellow for warm,
and red for hot:
38. // SensorHeatmapExtension.ts
(getColorFromValue method)
39. private getColorFromValue(value: number):
string {
40.     if (value <= 20) return "#0000FF"; //
Blue (Cold)
41.     if (value <= 23) return "#00FF00"; //
Green (Moderate)
42.     if (value <= 24) return "#FFFF00"; //
Yellow (Warm)
43.     return "#FF0000"; // Red (Hot)
44. }
```

2) Mover Implementation

```
1. // MoverPositionExtension.ts (moveMover
method)
2. private moveMover(targetPosition: number) {
3.     if (!this.moverGroup) {
5.         return;
6.     }
10.     const speed = 2; // Speed in units per
second
11.     const startPosition =
this.currentPosition;
12.     const distance =
Math.abs(targetPosition - startPosition);
13.     const duration = (distance / speed) *
1000; // Convert to milliseconds
17.     const startTime = performance.now();
19.     const animate = (time: number) => {
20.         const elapsed = time - startTime;
21.         const progress = Math.min(elapsed /
duration, 1); // Normalize progress (0 to 1)
23.         if (this.moverGroup) {
24.             this.moverGroup.position.y =
startPosition + (targetPosition -
startPosition) * progress;
25.         }
27.         if (progress < 1) {
28.             requestAnimationFrame(animate);
29.         } else {
30.             this.currentPosition =
targetPosition;
31.             console.log("☑ Mover reached
the designated floor.");
33.             // Play Beep Sound
34.             if (this.beepSound) {
35.                 this.beepSound.play();
37.             }
38.         }
39.     };
41.     requestAnimationFrame(animate);
```

```
42. }
44. // The extension also includes audio feedback when the mover reaches its destination:
45. // MoverPositionExtension.ts (loadBeepSound method)
46. private loadBeepSound() {
47.     if (!this.world) return;
49.     const listener = new THREE.AudioListener();
50.     this.world.camera.add(listener);
52.     this.beepSound = new THREE.Audio(listener);
54.  this.audioLoader.load("./sounds/elevator-ding-at-arenco-tower-dubai-38520.mp3", (buffer) => {
55.         this.beepSound!.setBuffer(buffer);
56.         this.beepSound!.setLoop(false);
57.         this.beepSound!.setVolume(1);
59.     });
60. }
```

G. Object Wrapper Implementation

```
1. // Object3DWrapper.ts (move method)
2. move (moveInfo: IMoveInfo): void {
3.     // Arrays are already initialized in property declarations
4.     if (moveInfo.repeatable !== true && this.checkMoveName(moveInfo)) {
6.         return
7.     }
8.     this.cancelMove()
9.     const oldPosition = this.object3D.position.clone()
10.     const oldRotation = this.object3D.rotation.clone()
11.     const oldLocation = {
12.         position: oldPosition,
13.         rotation: oldRotation
14.     }
15. this.beforeMoveLocation.push(oldLocation)
16.     this.moveHistoryList.push(moveInfo)
18.     const endPositionX: number = moveInfo?.position?.x === undefined ? oldPosition.x : oldPosition.x + moveInfo.position.x
19.     const endPositionY: number = moveInfo?.position?.y === undefined ? oldPosition.y : oldPosition.y + moveInfo.position.y
20.     const endPositionZ: number = moveInfo?.position?.z === undefined ? oldPosition.z : oldPosition.z + moveInfo.position.z
22.     const endRotationX: number = moveInfo?.rotation?.x === undefined ? oldRotation.x : oldRotation.x + degToRad(moveInfo.rotation.x)
23.     const endRotationY: number = moveInfo?.rotation?.y === undefined ? oldRotation.y : oldRotation.y + degToRad(moveInfo.rotation.y)
24.     const endRotationZ: number = moveInfo?.rotation?.z === undefined ? oldRotation.z : oldRotation.z + degToRad(moveInfo.rotation.z)
26.     const endPosition = new Vector3(endPositionX, endPositionY, endPositionZ)
27.     const endRotation = new Euler(endRotationX, endRotationY, endRotationZ)
28.     const endLocation = {
29.         position: endPosition,
30.         rotation: endRotation
31.     }
32.     const startLocation = {
```

```
33.         position:
oldLocation.position.clone(),
34.         rotation:
oldLocation.rotation.clone()
35.     }
36.     this.moveAction(startLocation,
endLocation, moveInfo?.duration === undefined ?
1000 : moveInfo.duration)
37. }
39. // The wrapper uses tweens for smooth
animations:
40. // Object3DWrapper.ts (moveAction method)
41. moveAction (startLocation: ILocationInfo,
endLocation: ILocationInfo, duration: number):
void {
42.     this.moveTween = new
Tween(startLocation)
43.         .to(endLocation, duration)
44.         .easing(Easing.Quadratic.Out)
45.         .onUpdate(() => {
46.   const position = startLocation.position
47.   const rotation = startLocation.rotation
48. this.object3D.position.set(position.x,
position.y, position.z)
49.  this.object3D.rotation.set(rotation.x,
rotation.y, rotation.z)
50.         })
51.         .onStop(() => {
52.             this.moveTween = null
53.         })
54.         .onComplete(() => {
55.             this.moveTween = null
56.         })
57.         .start()
59.     const animate = (time:
DOMHighResTimeStamp): void => {
60.         if (this.moveTween !== null) {
61.             requestAnimationFrame(animate)
62.             this.moveTween.update(time)
63.         }
64.     }
65.     animate(0)
66. }
```

H. Raycaster Implementation

```
 1. // GeniusRaycaster.ts (getIntersectedObject
method)
 2. private getIntersectedObject(event:
MouseEvent, container: HTMLElement): Object3D |
null {
 3.     const rect =
container.getBoundingClientRect();
 4.     this.mouse.x = ((event.clientX -
rect.left) / rect.width) * 2 - 1;
 5.     this.mouse.y = -((event.clientY -
rect.top) / rect.height) * 2 + 1;
 6.
 7.  this.raycaster.setFromCamera(this.mouse,
this.camera);
 8.     const intersects:
Intersection<Object3D>[] =
this.raycaster.intersectObjects(this.objects,
true);
10.     if (intersects.length > 0) {
12.         return intersects[0].object;
13.     }
15.     return null;
16. }
18. // The raycaster handles both mesh and
sprite objects, applying different visual
effects for each:
19. // GeniusRaycaster.ts (onHover method)
21. private onHover(event: MouseEvent,
container: HTMLElement): void {
22.     const intersectedObject =
this.getIntersectedObject(event, container);
intersectedObject?.type, intersectedObject);
```

```
25.     if (!intersectedObject) {
26.         this.onMouseLeave();
27.         return;
28.     }
30.     if
(this.selectedObjects.has(intersectedObject)) {
31.         return;
32.     }
34.     if (this.hoveredObject
&& !this.selectedObjects.has(this.hoveredObject
)) {
35.         if (this.hoveredObject instanceof
Mesh) {
36.             this.hoveredObject.material =
this.hoveredObject.userData.originalMaterial;
37.         } else if (this.hoveredObject
instanceof Sprite) {
38.             const spriteMaterial =
this.hoveredObject.material as SpriteMaterial;
39.             spriteMaterial.opacity = 1.0;
// Restore opacity for sprites
40.         }
41.     }
43.    if (intersectedObject instanceof Mesh) {
44. intersectedObject.userData.originalMaterial
= intersectedObject.material;
45.         intersectedObject.material =
this.highlightMaterial;
46.     } else if (intersectedObject instanceof
Sprite) {
47.         const spriteMaterial =
intersectedObject.material as SpriteMaterial;
48.         spriteMaterial.opacity = 0.5; //
Make sprite slightly transparent on hover
49.     }
51.     this.hoveredObject = intersectedObject;
53.     eventBus.emit("hover",
intersectedObject);
54. }
55.
```

I. User Interface Components

1) Sensor Visualization Techniques

```
1. // SensorSpritesExtension.ts
(onSpriteHovered method)
2. private onSpriteHovered(object: any): void
{
3.     if (object instanceof Sprite) {
5.         // Adjust opacity for highlight
effect
6.         const spriteMaterial =
object.material as SpriteMaterial;
7.         spriteMaterial.opacity = 0.7;
8.         eventBus.emit("sensorHovered",
object.name);
9.     }
10. }
```

2) Interactive Elements

```
1. // SensorSpritesExtension.ts (onSpriteClicked method)
2. private onSpriteClicked(object: any): void {
3.     const sensorId =
[...this.sensorSprites.entries()].find(([_id, sprite]) =>
sprite.uuid === object.uuid)?.[0];
5.     if (sensorId) {
7.         eventBus.emit("sensorSelected", sensorId);
9.         // Scale up clicked sprite
10.         this.resetAllSpriteScales(); // Reset all first
11.         const sprite = this.sensorSprites.get(sensorId);
12.         if (sprite) {
13.             sprite.scale.set(0.5, 0.5, 0.5); // Make it
bigger
16.             // Move & zoom camera to sensor position
17.             const targetPosition =
sprite.position.clone().add(new Vector3(0, 2, 5)); // Offset
slightly
18.     this.world.camera.position.lerp(targetPosition, 0.8); //
Smooth transition
```

```
19.                 this.world.camera.lookAt(sprite.position); //
Look directly at sensor
20.         }
21.     }
22. }
```

3) Control Panels and Dashboards

```
 1. // SensorListPanel.ts
 2. export default class SensorListPanel
extends UIBasePanel {
 3.     private dataView: HistoricalDataView |
null = null;
 5.     constructor() {
 6.         super("sensor-list-panel", " Sensor
List", 600, 300);
9.         this.dataView = getSensorData();
11.         // Initialize Table with Columns
12.         this.initializeTable([
13.         {title: "Sensor_id", field: "id"},
14.         {title: "Sensor", field: "sensor"},
15.         {title: "Group", field: "group"},
16.         ]);
17. this.initializeOutsideClickListener();
18.         // Populate Initial Data
19.  this.populateTable(this.getSensorData());
20.     }
22.     private getSensorData() {
23.         if (!this.dataView) return [];
25.         const rows = [];
26.         for (const [sensorId, sensor] of
this.dataView.getSensors().entries()) {
28.             rows.push({
29.                 id: sensorId,
30.                 sensor: sensor.name,
31.                 group: sensor.groupName,
32.             });
33.         }
34.         return rows;
35.     }
37.     // Override Row Click Logic
38.     protected onRowClicked(row:
HTMLTableRowElement) {
39.         const sensorId =
row.dataset.sensorId;
40.         if (!sensorId) return;
42.         super.onRowClicked(row);
43.         eventBus.emit("sensorSelected",
sensorId);
44.     }
45. }
```

References

[1] E. W. F. Peterson, " The Role of Population in Economic Growth.," *SAGE Open,,* no. 74, 2017.

[2] E. K. N. T. D. M. Y. K. a. S. M. A. Onat, "Design and Implementation of a Linear Motor for Multicar Elevators," *IEEE/ASME Transactions on Mechatronics,* vol. vol.15, no. 5, pp. 675-693, 2010.

[3] B. T. C. C. Raymon van Dinter, "Predictive maintenance using digital twins: A systematic literature review,," *Information and Software Technology,,* vol. 151, 2022.

[4] Q. a. A. S. a. Y. T. a. A. M. Xu, "Uncertainty-aware transfer learning to evolve digital twins for industrial elevators," in *ESEC/FSE 2022: Proceedings of the 30th ACM Joint European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, New York, 2022.

[5] TK Elevator, "MULTI," 2021. [Online]. Available: https://www.tkelevator.com/uk-en/products/elevators/multi/. [Accessed 4 2025].

[6] TK Elevator, "MAX," [Online]. Available: https://www.tkelevator.com/global-en/products/digital-solutions/country-selector/. [Accessed 4 2025].

[7] Y. Bouabdallaoui, Z. Lafhaj, P. Yim, L. Ducoulombier and B. Bennadji, "Predictive Maintenance in Building Facilities: A Machine Learning-Based Approach.," *Sensors,* vol. 21, no. 4, 2021.

[8] V. Villa, B. Naticchia, G. Bruno, K. Aliev, P. Piantanida and D. Antonelli, " IoT Open-Source Architecture for the Maintenance of Building Facilities," *Sensors,* vol. 11, no. 12, 2021.

[9] M. D. Isaac Opeyemi Olalere, "Early fault detection of elevators using remote condition monitoring through IoT technology," *South African Journal of Industrial Engineering,* vol. 29, pp. 17-32, 2018.

[10] K. Wang, Z. Li, K. Nonomura, T. Yu, K. Sakaguchi and O. Hashash, "Smart Mobility Digital Twin Based Automated Vehicle Navigation System: A Proof of Concept," *IEEE Transactions on Intelligent Vehicles,,* vol. 9, no. 3, pp. 4348-4361, 2024.

[11] T. Sumitani, Social Entrepreneurship Starting from Scratch, 2010.

[12] S. R. R. Boschert, " Digital Twin—The Simulation Aspect.," *Mechatronic Futures.,* 2016.

[13] Z. F. C. D. a. C. B. A. Fuller, "Digital Twin: Enabling Technologies, Challenges and Open Research," *IEEE Access,* vol. 8, 2020.

[14] The Government Accountability Office (GAO), "DIGITAL TWINS—VIRTUAL MODELSOF PEOPLE AND OBJECTS," *Science, Technology Assessment, and analytics,* Vols. GAO-23-106453 Digital Twins, 2023.

[15] D. GELERNTER, Mirror Worlds: or the Day Software Puts the Universe in a Shoebox...How it Will Happen and What it Will Mean, Oxford University Press, 1991.

[16] NASA, "Apollo 13: Mission Details," 7 2023. [Online]. Available: https://www.nasa.gov/missions/apollo/apollo-13-mission-details/. [Accessed 4 2025].

[17] D. T. Consortium, "The Digital Twin Consortium," 2022. [Online]. Available:

https://www.digitaltwinconsortium.org/.

[18] Simio, "Role of Digital Twin Technology in Industry 4.0," 3 2025. [Online]. Available: https://www.simio.com/role-of-digital-twin-technology-in-industry-4-0/#:~:text=Digital%20twin%20technology%20is%20revolutionizing,to%20smarter%2C%20more%20efficient%20operations.. [Accessed 4 2025].

[19] AIMultiple, "15 Digital Twin Applications/ Use Cases by Industry in 2025," 5 2025. [Online]. Available: https://research.aimultiple.com/digital-twin-applications/. [Accessed 6 2025].

[20] F. Q. R. Viradia, " Digital Twin Technology in Healthcare: Applications, Challenges, and Future Insights," *The New World Foundation.*

[21] Hexagon, "2025 digital twin statistics," [Online]. Available: https://hexagon.com/resources/insights/digital-twin/statistics.

[22] Singapore Government, "Virtual Singapore: Smart Nation Initiative," [Online]. Available: https://www.smartnation.gov.sg/initiatives/Urban-Living/virtual-singapore..

[23] A. J. M. Cespedes-Cubides, "A review of building digital twins to improve energy efficiency in the building operational stage.," *Energy inform,* vol. 7, no. 11, 2024.

[24] F. Jiang, H. Xie, S. Gandla and S. Fei, "Transforming Hospital HVAC Design with BIM and Digital Twins: Addressing Real-Time Use Changes," *Sustainability,* vol. 17, 2025.

[25] A. S. P. L. X. Z. Q. L. S.L. Zhou, "A comprehensive review of the applications of machine learning for HVAC," *DeCarbon,* vol. 2, 2023.

[26] F. B. Y. H. G. D. I. V. Aya Nabil Sayed, "Enhancing building sustainability: A Digital Twin approach to energy efficiency and occupancy monitoring," *Energy and Buildings,* vol. 328, 2025.

[27] G. M. Azanaw, "Application of Digital Twin in Structural Health Monitoring of Civil Structures: A Systematic Literature Review Based on PRISMA," *Journal of Mechanical and Construction Engineering,* vol. 4, 2024.

[28] Z. Sun, S. Jayasinghe, A. Sidiq, F. Shahrivar, M. Mahmoodian and S. Setunge, " Approach Towards the Development of Digital Twin for Structural Health Monitoring of Civil Infrastructure: A Comprehensive Review.," *Sensors,* vol. 25, 2025.

[29] 4.0 Solutions, "What is the Unified Namespace (UNS)?," 4.0 Solutions, 2025. [Online]. Available: https://www.iiot.university/blog/what-is-uns%3F.

[30] MQTT.org, "MQTT: The Standard for IoT Messaging," [Online]. Available: https://mqtt.org/.

[31] A. S. a. W. Chan, "A study of linear PMSM driven ropeless elevators," Building Services Engineering Research and Technology," vol. 40, no. 3.

[32] L. Z. M. L. P. Z. a. X. Z. C. Qiu, "Elevator fault diagnosis based on digital twin and PINNs-e-RGCN," *Scientific Reports,* vol. 14.

[33] Autodesk, Statista, "Building the future," Autodesk, 2018.

[34] T. Staff, "50 Years Ago: How Simulators Saved Apollo 13," 13 5 2020. [Online]. Available: https://televue.com/televueopticstalk/2020/05/13/50-years-ago-how-simulators-saved-apollo-13/. [Accessed 3 7 2025].