

## Highlights

### **TwinArch: A Digital Twin Reference Architecture**

Alessandra Somma, Domenico Amalfitano, Alessandra De Benedictis, Patrizio Pelliccione

- Existing DT architectures are domain-specific, they are not documented with multi-views, and merge in the same view structural and dynamic elements.
- Practitioners reported challenges in applying current DT standards.
- TwinArch integrates literature elements, insights from practitioners and DT development platforms.
- TwinArch is documented using the Views and Beyond method aligned with ISO 42010.
- Completeness, usefulness, usability of TwinArch are validated by DT experts.

# TwinArch: A Digital Twin Reference Architecture

Alessandra Somma<sup>a</sup>, Domenico Amalfitano<sup>a</sup>, Alessandra De Benedictis<sup>a</sup> and Patrizio Pelliccione<sup>b</sup>

<sup>a</sup>University of Naples Federico II, Naples, 80125, Italy

<sup>b</sup>Gran Sasso Science Institute (GSSI), L'Aquila, 67100, Italy

## ARTICLE INFO

**Keywords:**  
Digital Twin  
Reference Architecture  
Views and Beyond  
Systematic Literature Review

## ABSTRACT

**Background.** Digital Twins (DTs) are dynamic virtual representations of physical systems, enabled by seamless, bidirectional communication between the physical and digital realms. Among the challenges impeding the widespread adoption of DTs is the absence of a universally accepted definition and a standardized DT Reference Architecture (RA). Existing state-of-the-art architectures remain largely domain-specific, primarily emphasizing aspects like modeling and simulation. Furthermore, they often combine structural and dynamic elements into unified, all-in-one diagrams, which adds to the ambiguity and confusion surrounding the concept of Digital Twins.

**Objective.** To address these challenges, this work aims to contribute a domain-independent, multi-view *Digital Twin Reference Architecture* that can help practitioners in architecting and engineering their DTs.

**Method.** We adopted the *design science* methodology, structured into three cycles: (i) an initial investigation conducting a Systematic Literature Review to identify key architectural elements, (ii) preliminary design refined via feedback from practitioners, and (iii) final artifact development, integrating knowledge from widely adopted DT development platforms and validated through an expert survey of 20 participants.

**Results.** The proposed Digital Twin Reference Architecture is named **TwinArch**. It is documented using the *Views and Beyond* methodology by the Software Engineering Institute. TwinArch website and replication package: <https://alessandrasomma28.github.io/twinarch/>.

**Conclusion.** TwinArch offers practitioners practical artifacts that can be utilized for designing and developing new DT systems across various domains. It enables customization and tailoring to specific use cases while also supporting the documentation of existing DT systems.

## 1. Introduction

**Digital Twins** (DTs) are dynamic virtual representations of physical systems, enabled by seamless, bidirectional communication between the physical and digital realms [1, 2]. Unlike traditional simulators, DTs are continuously updated with real-world data, supporting advanced functionalities like predictive maintenance, real-time monitoring, and system control [3, 4]. These capabilities have positioned DTs as a transformative technology finding applications across diverse fields such as manufacturing, aerospace, and automotive industries [4, 5].

Although the concept of Digital Twins has existed for nearly two decades, interest from both academia and industry has surged only recently, and a universally accepted definition has yet to be established [6]. Tao *et al.* [7] introduced a five-dimensional model for digital twins, defined as:  $DT = \{P_s, V_s, DD, S_s, CN\}$ , where  $P_s$  represents the physical space of real-world entities and interactions,  $V_s$  denotes the virtual space with digital replicas dynamically reflecting physical behavior,  $DD$  comprises real-time and historical data enriched with domain knowledge,  $S_s$  provides services such as monitoring and prediction [8], and  $CN$  ensures seamless integration across these dimensions.

Despite the growing adoption and potential benefits of Digital Twins, both academia and industry face substantial

challenges in unlocking their full potential. The inherent complexity of DT systems, coupled with high-development costs and time-intensive maintenance, remains a major barrier to broader adoption [5, 9, 10]. A significant contributing factor to these challenges is the absence of a software **Reference Architecture** (RA) to systematically guide the design, development, and maintenance of these software-intensive systems, regardless of their application domain [5, 11, 12].

A RA provides an abstraction of software components, their roles, and their interactions, serving as a template or blueprint for creating concrete software architectures. It encapsulates the essential characteristics of systems within a particular domain and offers a structured framework for designing new systems or enhancing existing ones [13]. The importance of RAs in software development, particularly for DT systems, is evidenced by initiatives led by standardization bodies. For instance, ISO is developing: (i) a standard for RAs in enterprises, systems, and software (ISO/IEC/IEEE CD 42042<sup>1</sup>), and (ii) a standard for a DT Reference Architecture (ISO/IEC AWI 30188<sup>2</sup>), which are both still in the early stages of development.

Indeed, the standardization process is lengthy and involves multiple stages, often spanning several years. Moreover, the complexity of standards and the absence of practical tools to facilitate their adoption often hinder their

<sup>1</sup>ISO/IEC/IEEE CD 42042 standard, “Enterprise, systems and software — Reference architectures”, <https://www.iso.org/standard/87310.html>

<sup>2</sup>ISO/IEC AWI 30188 standard, “Digital Twin — Reference architecture”, <https://www.iso.org/standard/53308.html>

 alessandra.somma@unina.it (A. Somma);  
domenico.amalfitano@unina.it (D. Amalfitano);  
alessandra.debenedictis@unina.it (A. De Benedictis);  
patrizio.pelliccione@gssi.it (P. Pelliccione)  
ORCID(s):

implementation in real-world projects. For example, the ISO 23247 standard<sup>3</sup>, which offers a reference architecture for Digital Twins in manufacturing [14], has faced criticism for its limited applicability due to its domain-specific nature [15, 16]. Additionally, it has been noted that the standard lacks critical components, such as those related to data management [17]. Even within the manufacturing sector, practitioners have reported challenges in applying the standard effectively, as highlighted in [18], largely due to the absence of supporting artifacts for DT instantiation.

Given the limitations and slow adoption of standardization efforts, the DT research community has proposed various reference and software architectures to address this gap. However, most of these proposals are either domain-specific or tailored to particular services [19, 20, 21, 22], which constrains their flexibility and reduces their broader applicability and usefulness across various DT domains. Additionally, many of these architectures rely on a single, unified diagram combining structural elements from different abstraction levels, often overlooking dynamic aspects [23]. These all-in-one design approaches contradict the ISO/IEC/IEEE 42010 standard<sup>4</sup>, which recommends to document architectures by using multiple *architectural views*.

Architectural views highlight specific subsets of system elements and their relationships, designed to address the concerns of particular stakeholders [24]. However, even when attempts are made to separate architectural concerns into distinct views, the resulting DT architectures lack a holistic perspective, instead focusing narrowly on specific aspects of Digital Twins, such as modeling and simulation capabilities [25]. This limitation, influenced in part by the subjective interpretation of what defines a DT, results in fragmented architectures that fail to integrate structural and dynamic elements comprehensively. Consequently, existing solutions tend to be highly customized and tailored to specific cases, highlighting the pressing need for a multi-view and domain-independent Reference Architecture for DTs.

The goal of this work is to identify a domain-independent, multi-view Digital Twin Reference Architecture that can help practitioners in architecting and engineering their DTs. The proposed *Digital Twin Reference Architecture* is called **TwinArch**, and it is organized in multiple views [24], following the *Views and Beyond* (V&B) methodology proposed by the Software Engineering Institute (SEI). TwinArch synthesizes architectural elements from existing DT architectures, integrates feedback from DT practitioners and incorporates insights from three widely adopted DT development platforms—*Eclipse Ditto*, *Azure Digital Twins* (ADT), and *FIWARE*.

TwinArch is designed to deliver scientifically robust and practical artifacts for researchers and practitioners engaged in the design and development of DT systems across diverse

domains. These artifacts serve a *dual purpose*: supporting the documentation of existing DT systems and guiding the creation of new ones. Practitioners can utilize TwinArch as a foundational framework, adapting and customizing it to meet the specific requirements of their use cases, while also leveraging the provided mapping between architectural elements and the software tools of the selected platforms for practical implementation.

To perform the study of this paper, we employed the *design science* methodology, an iterative and structured approach conducted over three cycles. In the first cycle, a Systematic Literature Review (SLR) was performed to analyze existing DT architectures. The second cycle involved developing an initial draft of TwinArch, which was then preliminarily validated by practitioners participating in the project supporting this work. In the third cycle, TwinArch was further refined by integrating knowledge from the three selected platforms, identified through a specific search for DT platform solutions.

TwinArch's completeness, usefulness, and perceived usability were evaluated through an online survey conducted with DT experts. The results indicated a broadly positive perception of TwinArch, with respondents affirming its completeness, usefulness, and usability. TwinArch was particularly praised for providing clear guidelines and facilitating communication among stakeholders, developers, and researchers. Statistical and practical significance tests further confirmed that TwinArch is well-suited to serve as a complete, useful and usable Reference Architecture for Digital Twins.

The reminder of this paper is organized as follows. Section 2 summarizes related works on DT architectures. Section 3 describes the design science methodology, while Section 4 presents the proposed TwinArch. Section 5 details the online survey results. Section 6 discusses challenges in Digital Twin architectures. Finally, Section 7 draws our conclusions and introduces future work.

## 2. Related Work

The growing interest in Digital Twin technology has led to significant efforts to define software architectures that support their design, development, and deployment, as demonstrated by the increasing body of research on the topic [5, 9, 26]. For instance, Bolender *et al.* [19] developed a model-driven DT architecture for the self-adaptive manufacturing by incorporating domain-specific modeling and case-based reasoning. Similarly, Maceas *et al.* [20] employed a domain-driven design methodology to structure DT systems in highly evolving environments. Boyes *et al.* [27] proposed an analysis framework to identify common functional characteristics of DTs, addressing ambiguities caused by varying DT definitions.

Layered patterns are widely used to architect DTs. For example, Redelinghuys *et al.* [28] introduced a six-layer architecture to facilitate seamless data and information exchange between cyberspace and the physical twin, inspired by Cyber-Physical Systems (CPSs). Steinmetz *et al.* [29]

<sup>3</sup>ISO 23247 standard, “Automation systems and integration — Digital twin framework for manufacturing”, available at: <https://www.iso.org/standard/75066.html>

<sup>4</sup>ISO/IEC/IEEE 42010:2022 standard, “Software, systems and enterprise — Architecture description”, available at: <https://www.iso.org/standard/74393.html>

defined key components for DT-based systems with varying levels of granularity, organizing these components into four layers to capture system concerns. Similarly, Malakuti *et al.* [25] proposed an abstract four-layer architecture pattern to integrate information from diverse sources into DTs.

Despite these contributions, many DT architectures rely on single-view approaches. As highlighted in [27], this often results in confusion surrounding the DT concept and improper use of model elements, where structural and dynamic aspects are mixed instead of being represented in distinct views, as recommended by the ISO 42010 standard [24, 30]. Although multi-view approaches are less common, notable exceptions exist. For instance, Van Dinter *et al.* [21] proposed a multi-view reference architecture for DT-based predictive maintenance systems, organizing views into user, structural, and layered perspectives. Similarly, Tao *et al.* [23] introduced the makeTwin architecture, specifying ten functional modules for rapid DT prototyping and deployment, instantiated in the manufacturing domain.

The ISO 23247 standard, published in 2021 [14], defines a DT reference architecture specifically for manufacturing. It includes an entity-based reference model with four main entities: (i) Device Communication (responsible for data collection and control of Observable Manufacturing Elements, such as physical assets), (ii) Digital Twin (focused on modeling, synchronization, and management), (iii) User (hosting applications that utilize DT services), and (iv) Cross-System (providing overarching functionalities like security and data translation) [31]. The standard also includes a functional view, which details Functional Entities (FEs) to specify the functionalities at each level of the reference model.

Several studies have adopted ISO 23247 to design DT software architectures. For example, Bong Kim *et al.* [32] proposed a DT architecture for additive manufacturing to address process variability and enhance quality assurance. Spaney *et al.* [33] presented a standard-based model-driven DT architecture for milling processes, Melo *et al.* [34] applied the ISO 23247 standard to develop a DT for automotive assembly lines, focusing on process precision. Wallner *et al.* [35] extended the standard to design a DT for flexible manufacturing cells, integrating lifecycle management, path planning, and collision detection to manage reconfigurations. Caiza *et al.* [36] implemented an immersive DT architecture using augmented reality for real-time monitoring and control.

While these studies demonstrate the applicability of ISO 23247 across manufacturing scenarios, challenges remain in extending its use to other domains. For instance, Ferko *et al.* [16] explored its application in battery systems. Similarly, Shtofenmakher *et al.* [15] attempted to tailor the standard for aerospace use, focusing on on-orbit collision avoidance. Despite these efforts, researchers have highlighted significant limitations in ISO 23247, including domain-specific constraints, perceived misalignments and lack of concrete tools supporting the instantiation of DTs. A recent industrial survey [18] underscored the practitioners' difficulties in adopting the standard. For instance, they noted the absence

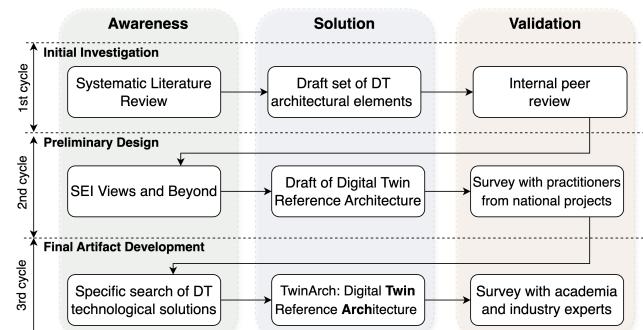
of important Functional Entities, such as those for data storage and management. This need is further validated by Kang *et al.* [17], who extended ISO 23247 with edge computing technologies to enhance data processing and decision-making in DT systems.

In line with the ISO 23247 standard, which is tailored to manufacturing, most of the DT architectures available in the literature are designed for specific domains or services, such as manufacturing [19], car-as-a-service [29], transportation [37], water treatment [38], and predictive maintenance [21]. This domain-specific focus limits their flexibility and applicability across diverse contexts, underscoring the need for a domain-independent and multi-view architecture. To address this challenge, the ISO organization is actively working on a DT reference architecture through initiatives like ISO 30188 (under development).

Summarizing, the discussed challenges and limitations of current DT architectures, highlight the need for a domain-independent, multi-view DT reference architecture. This study addresses these issues by: (i) documenting TwinArch using the Views and Beyond method in alignment with ISO 42010; (ii) integrating state-of-the-art architectural elements into TwinArch, informed by feedback from DT researchers and insights from three well-known DT development platforms; and (iii) offering reusable artifacts that can be applied by DT practitioners across multiple domains.

### 3. Methodology

Our work contributes to the definition of **TwinArch**, the *Digital Twin Reference Architecture*. To design TwinArch, we adopted the *design science* methodology, a structured process comprising three primary phases: *context awareness*, *solution synthesis*, and *solution validation* [39, 40]. As shown in Figure 1, which illustrates our methodology steps, we adopted an iterative process organized into three cycles, each encompassing the design science phases to understand the problem context, devise a solution, and validate it.



**Figure 1:** Overview of activities for designing TwinArch using the design science methodology.

**First Cycle: Initial investigation.** The first research cycle aimed to build awareness of the current state-of-the-art in Digital Twin architectures. To achieve this, a Systematic Literature Review was conducted to identify core Digital

Twins' architectural elements and uncover limitations in existing approaches, as detailed in Section 6. The identified elements were then refined and validated through internal peer reviews and collaborative discussions with co-authors, resulting in a consensus on the final set of elements and their responsibilities, forming a solid basis for the next research cycle.

**Second Cycle: Preliminary design.** In the second research cycle, the internally validated architectural elements formed the basis for the initial draft of the Digital Twin Reference Architecture. This draft was designed and documented using the SEI Views and Beyond method, which was analyzed during the context-awareness phase. The TwinArch draft and its elements were subsequently presented to five practitioners participating in one of the national projects supporting this study (see Sec. 8).

The preliminary evaluation involved two academic experts and three industry professionals working on Digital Twin case studies, ensuring a balanced assessment by combining theoretical insights with practical experience. Their diverse expertise provided valuable feedback, including recommendations to align the architectural elements to existing Digital Twin technological solutions and to incorporate detailed, practical examples. For instance, practitioners asked questions like, “*How does this align with platforms for DT development?*” and “*Can you provide examples from existing frameworks?*”. These suggestions were integrated into the final iteration.

**Third Cycle: Final Artifact Development.** The third cycle focused on refining TwinArch by integrating feedback and suggestions received during the previous iteration, and on performing a final validation through the collection of broader feedback from DT experts of academia and industry.

To address comments from practitioners during the previous cycle, a specific search of DT platforms and frameworks was carried out. Among available solutions, we selected Eclipse Ditto, Azure Digital Twins, and FIWARE as the reference platforms for our study due to their popularity in both industrial and research projects. This in-depth examination provided a comprehensive understanding of each platform's specific characteristics, enabling us to refine the architectural elements to better align with existing Digital Twin solutions and incorporate detailed, practical examples.

Finally, TwinArch was validated through an online survey involving 20 Digital Twin experts from both industry and academia to assess its completeness, usefulness, and perceived usability. These experts were identified through Digital Twin communities on platforms like LinkedIn and X (formerly Twitter), as well as professionals actively involved in DT-related projects. The feedback from the experts proved invaluable, identifying potential areas for future improvement, such as including support for domain-specific instantiation of TwinArch.

The rest of this Section provides details on the design science process focusing on the most relevant sub-phases. More specifically, Section 3.1 covers the initial investigation conducted through the literature review. Section 3.2 focuses

on the methodology adopted during the preliminary design iteration, centered around the Views and Beyond method. Section 3.3 refers to the final artifact development cycle and details the exploration of Digital Twin platforms and the online survey conducted for final validation.

### 3.1. Initial investigation: Building Awareness with Literature Review

We conducted a Systematic Literature Review based on the guidelines provided by Petersen *et al.* [41]. To ensure clarity and rigor, we defined a precise review protocol, defining the research goals, following the structured review process, and extracting data, while implementing measures to mitigate potential threats to the validity of the results. Further details can be found in the replication package.

**Review Process.** Our literature review began by defining research questions, from which a list of terms, synonyms and abbreviations was compiled. Following the guidelines of Kitchenham and Charters [42], a search string was constructed using the conjunction (AND) of disjunctions (OR) of the selected terms. The finalized search string is shown in the following box:

(“Digital Twin” OR “Virtual Twin” OR “Digital Replica” OR “Virtual Replica”) AND (Architect\* OR Framework OR Platform OR Document\* OR View OR Style)

Figure 2 illustrates the steps of the selection process, beginning with the execution of the search string in the Scopus database<sup>5</sup>. The inclusion criteria ensured that only studies directly related to the definition and documentation of Digital Twin architectures, written in English, peer-reviewed, and published in high-ranked journals or conference proceedings, were considered. The exclusion criteria, on the other hand, filtered out earlier versions of studies, publications conflating Digital Twin concepts with the Metaverse or AI models, and works that treated Digital Twins solely as simulated models. To minimize the risk of overlooking relevant literature, we conducted backward snowballing using Scopus and forward snowballing with Google Scholar<sup>6</sup>. This process resulted in a final set of 45 primary studies.

**Data Extraction.** The research goal and the screening phase guided the development of a data extraction scheme comprising a set of categories for collecting information from the selected studies. In addition to capturing the metadata of the publications, the scheme includes elements such as the number and type of architectural views and the notations used for documenting these views. Using this extraction scheme, data were systematically retrieved by analyzing the selected studies to gather relevant information.

**Results.** The complete SLR process along with the obtained results are reported in the replication package available at <https://alessandrasomma28.github.io/twinarch/slr.html>. In

<sup>5</sup>The search was conducted in June 2024.

<sup>6</sup><https://scholar.google.com/>

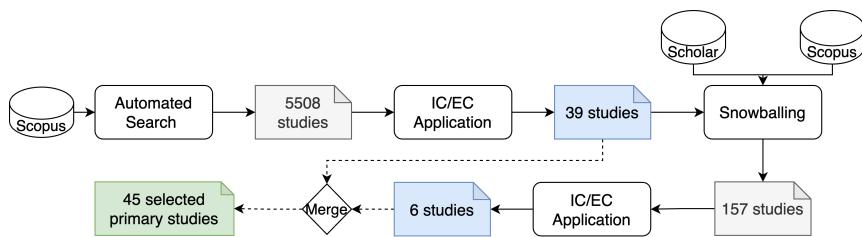


Figure 2: Systematic Literature Review process.

summary, the review revealed that only four out of the 45 selected papers presented more than one architectural view, and that the majority of studies relied on informal notations for architecture documentation. Moreover, from the analysis of selected papers we were able to identify some recurrent architectural elements that were used as the basis to build a first draft of the reference architecture (see Table 2 and Table 3).

### 3.2. Preliminary Design: Drafting TwinArch with the Views and Beyond

The initial draft of TwinArch was designed in the second cycle in accordance with the Views and Beyond method, a widely recognized approach for documenting architectures proposed by the Software Engineering Institute. Unlike fixed-view methods such as the Rational Unified Process, which relies on Krutchen' 4+1 model [43], the V&B method prioritizes flexibility, allowing architects to tailor views to the specific concerns and requirements of a given system [44].

The V&B method organizes the architecture documentation into three view types [30]. The *Module View* captures the system's decomposition into software modules, each responsible for a cohesive set of functionalities. The *Component-and-Connector (C&C) View* focuses on the system's runtime structure, representing components (processing units) and connectors (interactions) to highlight operational properties. The *Allocation View* maps the architecture to its physical or organizational environment, illustrating relationships between software and non-software elements, such as hardware or organizational structures.

Each view is defined by an architectural style, which specifies the types of elements, their relationships, and constraints on their usage [24, 30]. The method supports various notations for documenting architectures, ranging from informal to semi-formal (e.g., Unified Modeling Language, UML) and formal notations (e.g., ArchiMate). In addition to these core views, the V&B method incorporates supplementary documentation, referred to as *Beyond* aspects, such as behavioral views to capture dynamic interactions between architectural elements.

In this work, we document TwinArch using the module and component views, to describe the structural elements of a Digital Twin system. The Allocation View is excluded as TwinArch is a domain-independent reference architecture and does not include deployment details specific

to particular application domains [13]. Each view follows the architectural styles recommended by the V&B method. Furthermore, TwinArch includes behavioral documentation to describe dynamic interactions between elements within each view and traceability across views to ensure coherence in representing the overall Digital Twin system.

### 3.3. Final Artifact Development: TwinArch Refinement and Online Survey

TwinArch draft was refined by incorporating knowledge from three selected Digital Twin development platforms, as outlined in Subsection 3.3.1. The finalized TwinArch was validated through an online survey, with the process detailed in Subsection 3.3.2.

#### 3.3.1. Specific Search of DT solutions

In the third cycle, a targeted search for Digital Twin solutions was carried out, focusing on practical, open-source or commercial, widely-used platforms and frameworks, i.e. the collections of software tools designed to facilitate the creation, deployment, and maintenance of Digital Twins [45]. This search was independently conducted by two of the four authors, with the results and insights thoroughly discussed to reach consensus on refining the architectural elements and finalizing the design of TwinArch.

The search process was carried out using three primary channels: the Google search engine for a broad overview of available platforms; Google Scholar to examine academic discussions and analyses literature on DT technologies; and GitHub repositories and websites to identify open-source projects with active development and community support. A recent survey by Gil *et al.* [12] provided a useful starting point by analyzing 14 open-source frameworks and highlighting their varied approaches to offer DT-based services.

For instance, tools like *Eclipse Ditto* are well-suited for IoT-driven applications, while domain-specific platforms such as the *Digital Twin Cities Centre Platform (DTCC)* focus on smart city planning, and *CPS Twinning* supports cybersecurity focused Digital Twins. These findings emphasize the diversity of DT solutions and their alignment with specific use cases. Building on these insights, additional platforms identified during the search included *Azure Digital Twins* and *FIWARE* [12, 45, 46, 47, 48]. The final selected platforms (*Eclipse Ditto*, *Azure Digital Twins* and *FIWARE*) were chosen for their representation of different categories of Digital Twin solutions, balancing openness, functionality, and domain-specific applicability.

**Eclipse Ditto**<sup>7</sup> is a free platform developed as part of the Eclipse Internet of Things initiative, aimed at enabling the creation and management of DTs. It abstracts physical devices into faithful digital representations (called *Things*) and provides standardized Application Programming Interfaces (APIs) to allow seamless interaction with these virtual counterparts. Furthermore, Eclipse Ditto includes a *Gateway* component that facilitates external communication by supporting standard protocols such as MQTT and AMQP, along with the dedicated *Ditto Protocol*. This protocol employs JSON as the message format, enabling Eclipse Ditto to promote interoperability and simplify integration with external systems and services.

**Azure Digital Twins**<sup>8</sup> is a cloud-based Platform-as-a-Service solution that provides pay-as-you-go services, enabling the creation of digital models of physical environments. Built on the *Digital Twin Definition Language* (DTDL), a JSON-LD-based schema used for defining Digital Twins, ADT provides a framework for modeling physical entities, their properties, and relationships. Digital Twins are represented as *twin graphs*, dynamic graph-based models that capture the entities and relationships defined using DTDL. Moreover, the ADT platform integrates seamlessly with the broader Azure ecosystem. Data from IoT devices are ingested through *Azure IoT Hub*. Azure DT also connects with downstream services for analytics, storage, and visualization, such as *Azure Stream Analytics* for telemetry analysis, *Azure Data Lake* for long-term data storage, and *Azure Synapse Analytics* for advanced machine learning workflows.

**FIWARE**<sup>9</sup> is a free and open-source platform designed to facilitate the development of smart applications across a variety of domains, including smart cities, agriculture, and industry. It offers a modular architecture based on reusable and configurable software components called *Generic Enablers* (GEs), which communicate using the standardized *Next Generation Service Interface* (NGSI) protocol. At the heart of every FIWARE-based solution is the *Context Broker*, which manages real-time context data representing the state of physical and digital entities. Moreover, FIWARE offers tools such as *IoT Agents* that facilitate seamless integration with IoT devices by converting native protocols into NGSI format, and FIWARE *Cosmos* for integration with data processing and visualization frameworks.

FIWARE also spearheads the *Smart Data Models* initiative, which defines domain-agnostic JSON schemas to standardize data structures for smart applications. These models enhance interoperability across systems and platforms, addressing domains such as smart cities, environments, sensors, and agriculture<sup>10</sup>. The Digital Twin Definition Language used in Azure is built upon FIWARE data models.

<sup>7</sup><https://eclipse.dev/ditto/>

<sup>8</sup><https://azure.microsoft.com/en-us/products/digital-twins/>

<sup>9</sup><https://www.fiware.org/>

<sup>10</sup><https://github.com/smart-data-models/data-models>

**Table 1**

Summary of representative profiles.

ID	Experience	Affiliation	Role	# Individuals
P1	1 year	Industry	DT Developer	2
P2	1 year	Academia	Assistant Professor	1
P3	1-3 years	Industry	Research Engineer	2
P4	1-3 years	Academia	Researcher/Assistant Professor	6
P5	>3 years	Industry	(Senior) Research Engineer	5
P6	>3 years	Academia	Associate/Full Professor	4

### 3.3.2. TwinArch Online Survey

TwinArch was validated through an online survey which involved Digital Twin experts recruited from academia and industry. The objective of the survey was to gather practitioner feedback on the three quality factors—completeness, usefulness, and perceived usability—of TwinArch. In line with the Cruzes *et al.* guidelines [49], the online survey conducting process comprised four phases: (i) subject selection, (ii) questionnaire design, (iii) results analysis, and (iv) data reporting (see Section 5).

**Subject Selection.** TwinArch was created to assist practitioners in designing, developing and documenting DT systems across various domains. To evaluate its completeness, usefulness and perceived usability, survey participants were identified through three main sources: (i) authorship and contact details from the papers referenced during the TwinArch design, (ii) advertisements on social media and forums, and (iii) recognized experts in Digital Twin research. To minimize bias, outreach messages focused on the general objective of developing a reference architecture to support DT system design, without revealing specific details of the proposal. Recruitment took place from September 2024 to January 2025 and concluded when no further responses were received. A total of 20 *DT experts* participated in the survey, consisting of 9 industry practitioners and 11 academic researchers.

To gather responses, we sent out 546 emails: 367 were directed to DT developers, identified either through works with industrial affiliations or during the specific search for DT platforms, while 179 were sent to paper authors. The questionnaire was designed to evaluate the three quality factors for each view. Additionally, respondents were asked two supplementary questions: (i) how long they have worked in the DT field, and (ii) which DT development platforms they have adopted, if any. Based on their answers, we synthesized six representative profiles, as detailed in Table 1.

**Questionnaire Design.** The questionnaire was organized into 6 sections, comprising a total of 23 questions with a mix of closed and open-ended formats. The first section gathered background information about the respondents, while sections two through five focused on TwinArch’s architectural views. Each of these sections featured a closed question with a rating scale from Strongly Disagree to Strongly Agree to assess three key quality factors for each view. The final section assessed the overall TwinArch proposal and included open-ended questions, allowing participants to elaborate on their responses and provide additional insights, including potential strengths and limitations they identified.

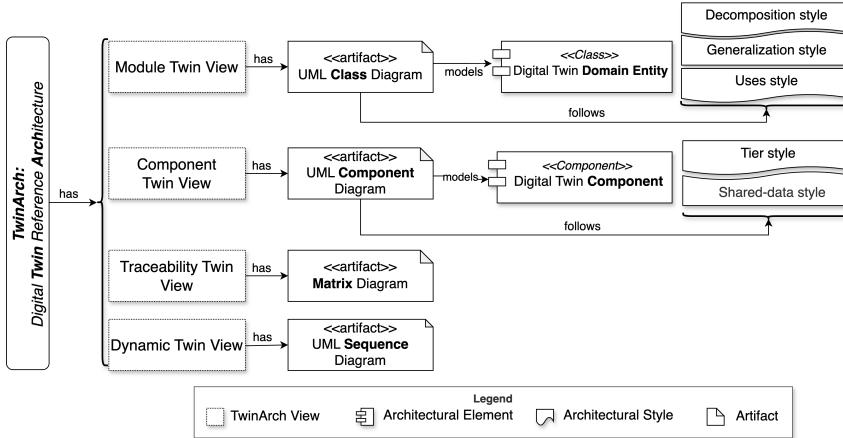


Figure 3: TwinArch Structure.

**Result Analysis.** The data were analyzed using quantitative methods, with statistical tests employed to enhance confidence and provide deeper insights. Specifically, Likert scale responses were visualized using Likert plots and further examined through statistical techniques, including box plots and significance testing. The survey results are detailed in Section 5.

## 4. TwinArch

TwinArch incorporates the architectural elements identified through the literature review, further refined based on insights from project researchers and mapped onto the three selected DT platforms. Designed for practitioners and researchers involved in the design and development of DTs, TwinArch offers scientifically sound and practical UML artifacts that can be customized to support the instantiation and implementation of new DTs in specific domains or serve as a guideline for documenting existing DTs.

The remainder of this Section is organized as follows. Section 4.1 provides an overview of TwinArch's structure. Sections 4.2 and 4.3 delve into the module and component views of TwinArch, respectively. Section 4.4 outlines the traceability view, linking the architectural elements of the aforementioned views. Lastly, Section 4.5 presents the dynamic view of two use cases, i.e., state monitoring and prediction. The complete architecture documentation is accessible on the TwinArch website: <https://alessandrasomma28.github.io/twinarch/>.

### 4.1. Structure Overview

Figure 3 presents an overview of the TwinArch's structure. In line with the SEI Views and Beyond, the proposed reference architecture is organized into multiple views, each addressing specific aspects of the DT system. The **Module Twin View** (MTV) models the domain entities of DTs using the *UML Class Diagram* notation. It employs decomposition, generalization, and usage styles to define high-level relationships and dependencies among domain classes. At a more detailed level, the **Component Twin View** (CTV)

focuses on the specific components of DTs and their interactions, represented through a *UML Component Diagram*. This view adopts tier-based and shared-data architectural styles to describe the relationships among components, providing a finer-grained representation than the MTV.

The **Traceability Twin View** (TTV) establishes a mapping between the structural elements of the MTV and CTV using a *Matrix Diagram*. This ensures a traceability path from the high-level domain entities defined in the MTV to the detailed components described in the CTV. Lastly, the **Dynamic Twin View** (DTV) employs *UML Sequence Diagram* to illustrate the interactions among structural elements (classes or components) at runtime. It provides a dynamic perspective, capturing interactions necessary to fulfill DT functionalities in two distinct use cases, i.e., physical system state monitoring and prediction, which represent two of the most typical key objectives of DTs across various industries.

### 4.2. Module Twin View

The *Module Twin View* defines the structure of a DT system by organizing it into modules (i.e., entities) and relationships between them. It can be expressed as:

$$MTV = \{DTE, ER\} \quad (1)$$

where:

- **DTE** represents the set of **Digital Twin Domain Entities**, which encapsulate the structural and functional characteristics of specific domain elements, such as physical twin or digital models, forming the foundational elements for building a DT system. The final set of DTEs, identified through the systematic review and refined through the specific search conducted on available DT platforms and solutions, is summarized in Table 2.

The table provides DTE's IDs, names, descriptions and the literature references from which they have been derived. Moreover, it reports the mapping of each DTE onto the corresponding software elements belonging to the DT platforms selected as reference for this study, namely FIWARE, Eclipse Ditto and Azure Digital Twins. In the

**Table 2**

Architectural elements of the Module Twin View: Digital Twin Domain Entities catalog.

ID	Name	Description	Literature Ref.	Azure Digital Twins	Eclipse Ditto	FIWARE
$dte_1$	PhysicalTwin	A real entity to be digitally replicated.	[10, 21, 22, 28, 37, 50, 51, 52, 53, 54, 55, 56, 57]	N/A	N/A	N/A
$dte_2$	DataProvider	A facilitator of data, responsible for transmitting raw data from the physical system to the DT.	[8, 10, 21, 22, 25, 28, 38, 47, 50, 51, 52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67]	N/A	N/A	N/A
$dte_3$	DataReceiver	A mediator between physical and digital twins, responsible for ensuring the transmission of feedback from the DT to the physical world.	[22, 25, 28, 53, 54, 56, 59, 62, 63, 64, 65]	N/A	N/A	N/A
$dte_4$	Adapter	An information converter, responsible for ensuring compatibility and integration between multiple data sources and the DT.	[8, 21, 28, 29, 50, 54, 58, 60, 62, 63, 65, 67, 68]	Event Routing (✓)	Protocol Adapter (✓)	IoT Agent (✓)
$dte_5$	P2DAdapter	An adapter for physical data, responsible for converting and preparing data for integration into the DT system.	[38, 50, 58, 62, 63, 65, 67]	Event Route (✓)	Connectivity API (✓)	IoT Agent (✓)
$dte_6$	D2PAdapter	An adapter for DT data, responsible for converting and preparing data for integration into $dte_1$ .	[58, 62, 63, 65, 67]	Event Grid (✓)	Connectivity API (✓)	IoT Agent (✓)
$dte_7$	DigitalRepresentation	A digital representation of a real-world entity, responsible for abstracting its key structural and behavioral aspects.	[21, 22, 23, 25, 27, 28, 29, 37, 38, 47, 50, 53, 57, 58, 59, 61, 66, 67, 68, 69, 70, 71, 72]	~	~	~
$dte_8$	DigitalShadow	A collection of temporal data traces, responsible for representing $dte_1$ states grouped by shadow types.	[8, 23, 50, 62, 63, 73]	Digital Twin Model (✓)	Things (✓)	Context Entities (✓)
$dte_9$	ShadowManager	A creator and manager of multiple $dte_8$ , responsible for lifecycle management of digital shadows.	[8, 23, 50, 53, 62, 63, 73]	Model Management (✓)	Thing Management (✓)	Context Broker (✓)
$dte_{10}$	DigitalModel	A digital representation of $dte_1$ behavioral aspects, for enabling dynamic simulation.	[21, 22, 23, 25, 27, 47, 50, 51, 52, 53, 60, 61, 67, 70, 71, 73]	X	X	X
$dte_{11}$	ModelManager	A creator and manager of multiple $dte_{10}$ , responsible for integrating and synchronizing multiple digital models.	[23, 50, 51, 53, 67, 73]	X	X	X
$dte_{12}$	TwinManager	A central orchestrator to $dte_9$ and $dte_{11}$ combined functionalities, responsible for cohesive management.	[10, 23, 25, 50, 51, 52, 53, 72, 73]	Model Management (~)	Thing Management (~)	Context Broker (~)
$dte_{13}$	ServiceManager	A creator of DT services, responsible for managing and executing DT services.	[10, 21, 22, 23, 25, 27, 28, 38, 50, 51, 52, 55, 58, 62, 71, 72, 74]	Azure Stream Analytics (✓)	Event Handling (~)	Perseo (~)
$dte_{14}$	FeedbackProvider	A generator of alerts, events, and commands, responsible for channeling feedback from the DT to the $dte_1$ .	[22, 29, 51, 53, 54, 56, 59, 64, 68, 73]	X	X	X
$dte_{15}$	DataManager	An aggregator of data circulating within the DT, responsible for efficient management, storage, and retrieval.	[10, 21, 22, 23, 27, 29, 38, 47, 50, 51, 53, 54, 55, 56, 58, 60, 61, 62, 63, 68, 72, 75]	Azure Event Hub (✓)	Thing Management (~)	Context Broker and QuantumLeap (✓)
$dte_{16}$	DataModel	A model representing the logical structure of exchanged data, for ensuring data interoperability.	[27, 37, 58, 60, 62, 63, 64, 69, 71]	Interoperability DTDL Models (✓)	Thing Management (✓)	Smart Data Models (✓)

table, in particular, mappings are represented using three symbols: ✓, ~, and X. A ✓ symbol indicates that the DTE class can be fully implemented using the tools offered by the respective DT platform. A ~ symbol denotes that the DTE class is only partially implementable with the existing tools, necessitating additional resources to complete the element. Lastly, a X symbol signifies that the DTE class responsibilities cannot be addressed using the tools provided by the DT platform.

- ER denotes the set of **Entity Relationships** that describe connections among DT domain entities. These relationships are established based on the architectural styles adopted in the MTV design.

- ◊ The *Decomposition Style* uses a divide-and-conquer approach to manage the system's complexity by breaking it into smaller modules, introducing the *is-part-of* relationship. This relationship can either represent a strong composition, where the part cannot exist independently of the whole, or an aggregation, where the part can exist independently of the whole.
- ◊ The *Generalization Style* models common functionalities across modules to promote sharing and reuse, introducing the *is-a* relationship.
- ◊ The *Uses Style* models dependencies between modules, supporting incremental design and introducing the *use* relationship.

- ◊ The abstraction relationship is introduced to represent the connection between the domain entity representing the physical system and its corresponding virtual counterpart.

Figure 4 depicts the *UML Class Diagram* illustrating the Digital Twin Domain Entities as classes, along with their interrelationships. Use and abstraction relationships are depicted in the diagram through arrows to which the corresponding stereotypes are applied. As for the *is-part-of* relationship, composition is represented with a filled diamond at the end of the association line that connects to the whole, while aggregation is depicted with an empty diamond. Relationship multiplicity specifies how many instances of a part can be associated with a single instance of the whole. Finally, the generalization relationship is represented as a solid line with a hollow triangle arrowhead pointing towards the more general (parent) class.

The following paragraphs illustrate the architectural elements of the Module Twin View providing a description of their main functions and on their mutual relationships. Moreover, in yellow boxes the reader can find a discussion on whether and how each entity is mapped to the considered software technologies. When applicable, relevant examples of the mapping are also discussed.

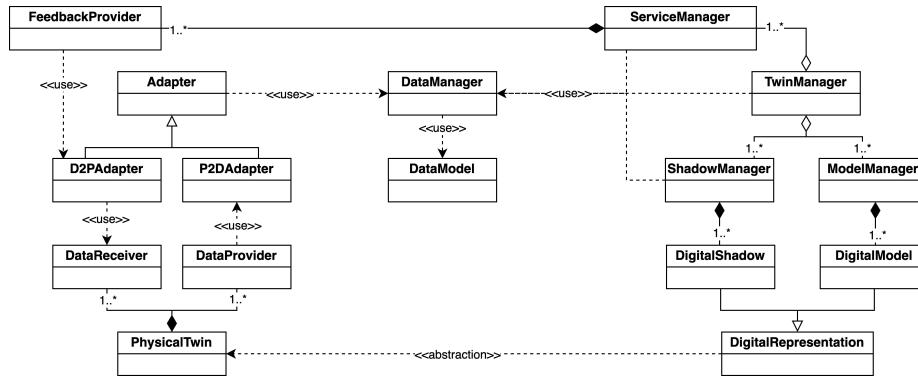


Figure 4: Module Twin View: UML Class Diagram.

### PhysicalTwin and DigitalRepresentation.

PhysicalTwin represents the real-world entity that is digitally replicated within the DT system. It acts as source of truth, providing essential data and state information to its virtual counterpart for various use cases, including simulation, monitoring, and prediction.

DigitalRepresentation, on the other hand, abstracts the structural and behavioral characteristics of PhysicalTwin, accommodating different levels of granularity. The abstraction relationship ensures that DigitalRepresentation captures the key properties and functionalities that are relevant to the objectives of the DT system. For example, for system monitoring, it may include structural and behavioral features like dimensions and states, while omitting irrelevant attributes such as aesthetic details. Additionally, abstraction allows DigitalRepresentation to incorporate derived or aggregated data, such as maintenance history or performance metrics, enhancing its value and utility.

As illustrated in Fig. 4 and discussed in detail later, DigitalRepresentation can be further classified into digital shadows and digital models.

PhysicalTwin exists solely in the physical domain and is not mapped to any platform, as the discussed technologies focus on digital aspects. In contrast, DigitalRepresentation is only partially supported, with the selected technologies addressing specific aspects of its specialized forms, such as shadowing entities, rather than fully covering the entire DigitalRepresentation.

### DataProvider, DataReceiver and Adapters.

DataProvider acts as an intermediary between the physical and digital twins, generating the flow of information from the physical world into the DT system. It ensures that data from the PhysicalTwin are effectively transmitted to the digital counterpart. DataReceiver operates in the opposite direction, mediating the flow of information and feedback from the DT to the physical world. Together, DataProvider and DataReceiver maintain bidirectional synchronization between the physical and digital spaces. As depicted, DataProvider and DataReceiver are modelled as a-part-of PhysicalTwin by a strong composition relationship,

meaning that the lifetimes of DataProvider and DataReceiver are encompassed within the lifetime of PhysicalTwin.

Adapter performs the transformations necessary for seamless data exchange between multiple and heterogeneous data sources and the DT system. Adapter class is specialized into P2DAdapter and D2PAdapter, which manage data flows in specific directions. P2DAdapter focuses on physical-to-digital data flows, transforming and preparing data sent by DataProvider for integration into the DT system. Conversely, D2PAdapter handles digital-to-physical data flows, adapting and preparing feedback or commands provided by the DT system for DataReceiver.

As shown in Fig. 4, there is a usage relationship from DataProvider to P2DAdapter indicating that the former relies on the functions offered by the latter: in fact, data retrieved by DataProvider from PhysicalTwin are processed by P2DAdapter and transformed into a format compatible with the DT system, defined by DataModel class, prior to be transmitted to DataManager. A similar usage relationship holds from D2PAdapter to DataReceiver that involves an opposite data flow, omitted for brevity.

DataProvider and DataReceiver are not directly mapped to any of the selected platforms, as they represent physical-world sensors and actuators. Instead, P2DAdapter and D2PAdapter are supported by Event Routing tools in Azure Digital Twins, the Connectivity API in Eclipse Ditto, and the IoT Agents available in FIWARE.

**Exemplars in DT Platforms:** Let us consider a traffic loop sensor that measures vehicle flow and periodically transmits its readings to the Digital Twin system for integration into the digital model. Below, we examine how this sensor interacts with adapters within the Azure Digital Twins, Eclipse Ditto, and FIWARE platforms.

In **Eclipse Ditto**, P2DAdapter leverages the *Connectivity API*<sup>a</sup> to handle data streams from the traffic loop sensor. An example request to the Ditto Connectivity API is described below:

```

curl -X POST
'http://ditto/connectivity?filter=type=
  trafficLoop'
-d '{"input": "vehicleCount",
  "output": "processedFlow"}'
  
```

The filter ensures that only data from traffic loop sensors are processed. The payload specifies the transformation from raw vehicle count data to a processed flow representation. The adapter retrieves

the traffic flow data, enriches them with additional metadata (e.g., timestamp, location), and integrates them into the Ditto-managed Twin in a structured format.

In **Azure Digital Twin**, P2DAdapter is implemented using *Event Routes*<sup>b</sup>, which route telemetry data from the sensor to the target Twin. A sample configuration for an Event Route is as follows:

```
{ "id": "trafficToTwinRoute",
  "source": "/eventhub/telemetry",
  "target": "/digitalTwins/trafficTwin",
  "filter": "$event.properties.sensorType == 'trafficLoop'" }
```

This configuration defines the route's unique identifier, the telemetry data source, and the target Twin. A filter ensures only traffic loop data are processed. The traffic loop sensor sends vehicle flow data via Azure Event Hubs<sup>c</sup>. The Event Route processes the readings and transforms them into a JSON format compatible with the schema of the target Twin.

In **FIWARE**, P2DAdapter is implemented using an IoT Agent, such as *IoT Agent Ultralight 2.0*<sup>d</sup>, which converts lightweight payloads into NGSI-LD context updates. A sample interaction is the following:

```
curl -iX POST
'http://<iot-agent-host>:<port>/iot/d?k=<apikey>
  &i=<device-id>'
-H 'Content-Type: text/plain' -d 'f|35'
```

In this case, the IoT Agent firstly authenticates the device using the API key and the identifier and then translates the received payload (f|35, where f denotes traffic flow and 35 the number of vehicles measured) into an NGSI-LD compliant structure. This transformed context is forwarded to the FIWARE Context Broker, enabling it to store and process the sensor data within the DT system.

<sup>a</sup><https://eclipse.dev/ditto/basic-connections.html>  
<sup>b</sup><https://learn.microsoft.com/en-us/azure/digital-twins/concepts-route-events>  
<sup>c</sup><https://azure.microsoft.com/en-us/products/event-hubs>  
<sup>d</sup><https://github.com/telefonicaid/iotagent-ul>

## DigitalShadow and ShadowManager.

DigitalShadow is a specialization of DigitalRepresentation, focusing on data-related aspects of PhysicalTwin.

It represents a collection of temporal data traces that capture the states of PhysicalTwin over time. This data-centric representation is grouped by shadow types, enabling advanced functionalities such as anomaly detection, predictive maintenance, and historical analysis.

ShadowManager oversees the lifecycle of multiple Digital Shadow instances, including their creation, updates, and deletion. It organizes shadows by temporal properties and types, maintaining their integrity and consistency and ensuring seamless coordination across DT system elements. The relationship between ShadowManager and DigitalShadow is a strong composition (is-part-of) relationship, as the lifetime of the shadow instances is dependent on the lifetime of the Manager.

DigitalShadow and ShadowManager are both supported by the selected DT platforms. In Azure Digital Twins, DigitalShadow is

implemented using Digital Twin Model to store data traces, while model management tools support ShadowManager for lifecycle handling. Eclipse Ditto models DigitalShadow as Things and uses Thing Management for managing multiple shadows. FIWARE represents shadows using NGSI-LD Context Entities and employs the Context Broker to manage their lifecycle.

## DigitalModel and ModelManager.

DigitalModel is the digital representation of the behavioral aspects of PhysicalTwin. It focuses on modeling the operational behavior of the physical entity, enabling dynamic simulations and predictive analysis. While DigitalShadow captures the historical states of PhysicalTwin, DigitalModel complements this by simulating current and future states, offering insights into system performance under different conditions, and enabling scenario-based analysis.

ModelManager is responsible for overseeing and managing multiple DigitalModel instances. It facilitates the integration, synchronization, and orchestration of these models, ensuring cohesive and accurate behavioral simulations. Additionally, ModelManager ensures the consistency of the models, aligning them with the corresponding PhysicalTwin and maintaining their integrity within the broader DT system. For this reason, the relationship between ModelManager and DigitalModel is a strong composition (is-part-of) relationship.

DigitalModel and ModelManager are not natively supported by the selected platforms, as these primarily address the structural modeling of physical assets and lack features for behavioral modeling. Implementing these entities requires the integration of specialized simulation tools. For instance, MATLAB Simulink can model and simulate dynamic systems, while Eclipse SUMO (Simulation of Urban Mobility) is suitable for traffic and urban planning simulations. Other examples include AnyLogic for multi-method modeling and Python-based frameworks such as OpenModelica for system-level simulations.

## TwinManager.

TwinManager serves as the central orchestrator, managing and coordinating the functionalities of both ShadowManager and ModelManager. It ensures the cohesive operation of the DT system by aligning the data-driven aspects represented by digital shadows with the simulation-driven aspects captured by digital models. Additionally, TwinManager handles cross-functional tasks such as synchronizing data between shadows and models.

TwinManager is partially supported by the selected platforms because it manages both data-driven flows from ShadowManager, which are fully supported, and simulation-driven flows from ModelManager, which are not supported. Azure Digital Twins provides Twin Management for orchestrating operations, Eclipse Ditto offers Ditto Management for device interactions and shadow functionalities, and FIWARE uses the Context Broker for synchronizing and integrating services.

## ServiceManager and FeedbackProvider.

ServiceManager is responsible for implementing, coordinating, and managing the services offered by the DT system, such as monitoring, anomaly detection, and prediction. FeedbackProvider produces the DT feedback, including alerts, events, and commands, directly linked to the services managed by ServiceManager. The operations of both entities enable the DT to react to observed conditions, issue alerts, and send commands to influence the behavior of PhysicalTwin. As critical entities, they support closed-loop operations, providing real-time feedback to drive adaptive responses in the physical domain. The relationship between ServiceManager and FeedbackProvider is a strong composition (is-part-of) relationship.

ServiceManager is fully supported by Azure DTs, leveraging Azure Stream Analytics to handle service-related tasks effectively. In comparison, the other two platforms offer partial service management capabilities. Eclipse Ditto facilitates service orchestration through its management and integration features, including event handling, while FIWARE supports real-time alerts and actions using tools such as Perseo. FeedbackProvider is not natively supported on any of the selected platforms, necessitating custom implementation to develop specialized feedback generation mechanisms tailored to the implemented services.

### DataManager and DataModel.

DataManager is responsible for aggregating, managing, and retrieving data within the DT system, ensuring efficient storage, consistency, and integration. It relies on DataModel, which defines the logical structure of the data, facilitating interoperability by standardizing data formats for internal and external exchanges.

DataManager is a pivotal entity utilized by various DTEs. For instance, TwinManager uses it to manage data flows between shadows and models, ServiceManager depends on it for service-related data operations, and ShadowManager leverages it to store temporal data traces representing PhysicalTwin' states.

DataManager and DataModel are fully supported by the Azure Digital Twins and FIWARE platforms. Azure DTs utilizes the Digital Twin Definition Language for data modeling and Azure Event Hub to address critical data management functions. FIWARE employs Smart Data Models for data definitions and combines the Context Broker with QuantumLeap to facilitate efficient data storage and management. In contrast, Eclipse Ditto offers comprehensive support for data modeling through its Things Management API, which provides only partial support for data management functionalities.

**Exemplars in DT Platforms:** Let us consider the same traffic loop sensor measuring vehicle flow of the previous example. The exchanged measurement are modeled by the selected platforms as explained below.

In *Eclipse Ditto*, the *Things Management API*<sup>a</sup> is used to define and manage digital representations of physical devices (the Things). For example, a traffic loop sensor can be represented as a Thing with a unique identifier, thingId, and an attribute, vehicleCount, which stores the observed traffic flow as an integer value. The following JSON schema demonstrates how the sensor can be modeled:

```
{ "thingId": "example:TrafficSensor",
```

```
"attributes": {  
    "vehicleCount": {  
        "type": "integer",  
        "value": 35  
    } } }
```

In *Azure Digital Twins*, the *DTDL*<sup>b</sup> is adopted to define the structure of Digital Twin models. This language allows for creating detailed representations of physical entities, including their properties, telemetry, commands, and relationships. For instance, a traffic loop sensor can be modeled as an interface in DTDL, representing its data collection functionality, as illustrated below:

```
{ "@id": "dtmi:example:TrafficSensor;1",  
  "@type": "Interface",  
  "contents": [ {  
      "@type": "Telemetry",  
      "name": "vehicleCount",  
      "schema": "integer"  
    } ] }
```

Data collected by Azure Event Hub can be seamlessly stored and analyzed using Azure services such as Azure Data Lake or Cosmos DB, facilitating advanced data-driven insights and operations.

In *FIWARE*, *Smart Data Models*<sup>c</sup> are employed for data definition and the *Context Broker* for data management. Traffic loop sensor data can be represented using an NGSI-LD schema based on the Transportation data model. For instance, the sensor could be represented as an entity of type *TrafficFlowObserved*, with a unique identifier and attributes capturing its observations:

```
{ "id": "urn:ngsi-ld:TrafficFlowObserved:TLF01",  
  "type": "TrafficFlowObserved",  
  "location": {  
    "type": "Point",  
    "coordinates": [40.7128, -74.0060]  
  },  
  "vehicleFlow": {  
    "value": 35,  
    "observedAt": "2024-12-10T12:00:00Z"}}
```

<sup>a</sup><https://eclipse.dev/ditto/basic-thing.html>  
<sup>b</sup><https://azure.github.io/opendigitaltwins-dtdl/DTDL/v3/DTDL.v3.html>  
<sup>c</sup><https://github.com/smart-data-models>

### 4.3. Component Twin View

The *Component Twin View* models the architecture of a DT software system by specifying its components and their connectors. This view provides a finer level of granularity than the Module Twin View, as it emphasizes the internal software components and is more closely aligned with software implementation. The Component Twin View is expressed as:

$$CTV = \{ DTC, CR \} \quad (2)$$

where:

- *DTC* represents the set of **Digital Twin Components**, which are self-contained software entities encapsulating specific functionalities within the DT system. Each component is designed to perform a distinct role, such as data processing, state monitoring, or simulation. Similar to the Module Twin View, the final set of DTCs, identified

**Table 3**

Architectural elements of the Component Twin View: Digital Twin Components catalog.

ID	Name	Description	Literature Ref.	Azure Digital Twins	Eclipse Ditto	FIWARE
$dtc_1$	PhysicalTwin	A real-world asset to be replicated by the Digital Twin.	[8, 10, 19, 20, 22, 37, 50, 51, 52, 53, 56, 59, 64, 73, 74, 76, 77]	N/A	N/A	N/A
$dtc_2$	DataProvider	An intermediary facilitating the transmission of raw data from the physical to the Digital Twin.	[8, 10, 19, 20, 22, 37, 50, 51, 52, 53, 54, 56, 59, 64, 73, 74, 76, 77]	N/A	N/A	N/A
$dtc_3$	DataReceiver	A receiver ensuring feedback, updates, or commands from the $dtc_2$ reach the Physical Twin.	[58, 62, 63, 65, 67]	N/A	N/A	N/A
$dtc_4$	P2DAdapter	A converter that translates physical system data into formats usable by the DT.	[38, 50, 58, 62, 63, 65, 67]	Event Route (✓)	Connectivity API (✓)	IoT Agent (✓)
$dtc_5$	D2PAdapter	A translator that transforms Digital Twin outputs into formats usable by $dtc_1$ .	[58, 62, 63, 65, 67]	Event Grid (✓)	Connectivity API (✓)	IoT Agent (✓)
$dtc_6$	DataProcessor	A processing unit that filters and organizes raw data, preparing them for integration into the DT system.	[8, 10, 19, 22, 23, 27, 37, 50, 51, 52, 53, 56, 59, 60, 61, 64, 69, 70, 74, 76, 77, 78]	Azure Event Hub (✓)	Thing Management (✓)	Context Broker (✓)
$dtc_7$	StorageManager	A component that organizes and manages shared data repository $dtc_6$ for efficient storage and retrieval.	[8, 19, 21, 22, 38, 51, 53, 56, 58, 60, 62, 63, 68, 72, 76, 78]	Azure Event Hub (✓)	Thing Management (~)	Context Broker (✓)
$dtc_8$	DataManager	A centralized component ensuring data consistency and availability, aggregating $dtc_6$ and $dtc_7$ functionalities.	[8, 19, 21, 22, 37, 51, 53, 54, 62, 63, 64, 69, 76, 78]	Azure Event Hub (✓)	Thing Management (~)	QuantumLeap (~)
$dtc_9$	SharedStorage	A data accumulator, responsible for storing heterogeneous data from both the physical and digital twins.	[8, 19, 21, 22, 23, 37, 38, 50, 51, 53, 54, 56, 58, 60, 61, 62, 63, 64, 68, 69, 72, 74, 75, 76, 77, 78]	Azure Data Lake or Cosmos DB (✓)	MongoDB (~)	MongoDB, TimescaleDB, CrateDB (~)
$dtc_{10}$	ShadowManager	A component responsible for creating, managing, and overseeing the lifecycle of multiple digital shadows.	[8, 19, 23, 27, 51, 60, 62, 63, 64, 72, 73, 74, 78]	Model Management (✓)	Ditto Management (✓)	Context Broker (✓)
$dtc_{11}$	ModelManager	A component responsible for creating and managing multiple digital models to simulate different aspects.	[23, 50, 60, 73, 79]	X	X	X
$dtc_{12}$	ModelEngine	A processing unit of digital models, responsible for executing simulations and generating results based on the modeled scenarios.	[23, 50, 60, 73, 79]	X	X	X
$dtc_{13}$	Simulator	A virtualizer of real-world systems, responsible for simulating the behavior of the physical system under various conditions.	[20, 21, 22, 23, 27, 28, 29, 38, 47, 50, 51, 61, 64, 72]	X	X	X
$dtc_{14}$	TwinManager	An orchestrator synchronizing the functionalities of shadow and model managers with the service-related components for cohesive DT operations.	[8, 19, 53, 60, 64, 65, 76, 79, 80]	Twin Management (~)	Ditto Management (~)	Context Broker (~)
$dtc_{15}$	StateMonitor	A monitoring component, responsible for collecting and forwarding real/simulated states to other components for further analysis or action.	[20, 22, 50, 53, 56, 60, 64, 65, 70, 74, 78, 80]	Azure Stream Analytics (✓)	Event Handling (~)	FIWARE Cosmos (✓)
$dtc_{16}$	DeviationDetector	A comparison unit, responsible for identifying deviations by comparing real or predicted states with expected states to detect any deviation.	[8, 10, 21, 22, 59, 64, 65, 75, 80]	Multivariate Detection toolkit (✓)	X	FIWARE Perseo (~)
$dtc_{17}$	Predictor	A forecasting component, responsible for using current and historical data to anticipate potential future states of the physical system.	[8, 10, 21, 38, 54, 58, 59, 60, 62, 63, 79, 80]	Azure ML and DL (✓)	X	FIWARE Cosmos (~)
$dtc_{18}$	Analyzer	A detailed analytical component, responsible for analyzing real, predicted and simulated states to extract meaningful insights.	[8, 10, 19, 21, 22, 27, 29, 38, 47, 54, 56, 58, 59, 60, 64, 70, 72, 73, 74, 76, 77, 79, 80]	Azure Stream Analytics (✓)	~	FIWARE Perseo and Cosmos (~)
$dtc_{19}$	SolutionFinder	A component responsible for finding the best set of actions to return the system to a desired state after deviation detection.	[8, 19, 22, 59, 74, 79]	X	X	X
$dtc_{20}$	ScenarioGenerator	A generator designed to create diverse scenarios, facilitating the preparation and execution of multiple simulations.	[10, 23, 79, 80]	X	X	X
$dtc_{21}$	Planner	A planning unit, responsible for developing a solution plan to restore the system to a desired state when deviations or anomalies are detected	[19, 22, 59, 70, 73, 74]	X	X	X
$dtc_{22}$	FeedbackExecutor	A generator of alerts or actionable instructions into the physical system $dtc_1$ .	[8, 19, 22, 29, 53, 59, 64, 68, 70, 73, 74]	X	X	X

through the literature review and refined with the platforms specific search, is detailed in Table 3.

The table details DTCs identifiers, name, description, references from the literature. Moreover, as done in the Module Twin View, Tab. 3 reports the mapping onto the software tools of the selected technologies (Azure Digital Twins, Eclipse Ditto, and FIWARE) specifying whether the mapping is total (✓), partial (~), or not possible (X).

- CR denotes the set of **Component Relationships**, which describe the interactions and dependencies among components. These relationships are defined based on the architectural styles used in the design.

- ◊ The *Tier Style* organizes components into functional groups according to their execution structures, modeling their composition through the is-part-of relationship.

- ◊ The *Shared-Data Repository Style* defines the usage relationship, illustrating how components access a shared repository to read or write data.

Beyond the above mentioned relationships, the Component Twin View incorporates additional relationships commonly adopted in C&C views [30], namely:

- ◊ the assembly relationship, which connects a component's required interface to the provided interface of another component.
- ◊ the port attachment or association relationship, which models how components exchange information or collaborate, typically visualized as connections between their ports.
- ◊ the interface delegation relationship, which links a component's internal ports to its external ports in cases where the component includes a sub-architecture.

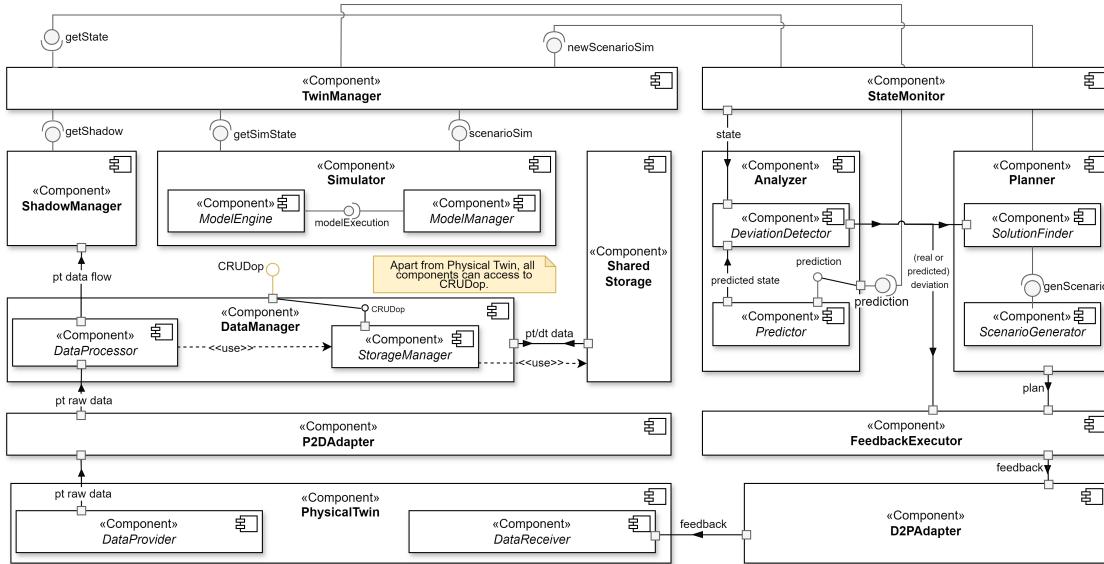


Figure 5: Component Twin View: UML Component Diagram.

Figure 5 presents the *UML Component Diagram*, illustrating the Digital Twin Components and their relationships. The subsequent paragraphs provide an in-depth explanation of the architectural elements of the Component Twin View, their relationships, and their mappings the entities of the module view. Where applicable, examples of usage of the components in concrete scenarios is also provided, together with specific examples of implementation within the considered platforms.

#### PhysicalTwin, DataProvider and DataReceiver.

PhysicalTwin, DataProvider and DataReceiver components directly map onto the respective entities of the MTV. In particular, DataProvider and DataReceiver are modeled as sub-components of the PhysicalTwin, enabling its interaction with P2DAdapter and D2PAdapter via suitable ports representing the flow of information.

#### P2DAdapter and D2PAdapter.

P2DAdapter and D2PAdapter components directly map onto the respective entities of the MTV. The former receives data from the DataProvider and transforms and prepares them before sending them to the specific components devoted to data management. The latter, dually, transforms information (typically commands) coming from the components responsible for feedback generation into a format compatible with the physical infrastructure before sending them to the DataReceiver.

#### DataManager, DataProcessor, StorageManager and SharedStorage.

Raw data provided by P2DAdapter are cleaned, filtered, and organized into a standardized format that aligns with the reference data model by the DataProcessor component. These standardized data are used by the ShadowManager to create and organize digital shadows based on predefined types,

and is fundamental to enable other critical operations such as feeding digital models and monitoring system behavior. By combining the historical data traces with simulation results, the system can achieve comprehensive analysis and predictive capabilities.

To support all above mentioned operations, the data are organized in a SharedStorage component: it acts as a passive component, serving as a central repository for securely storing processed data, digital shadows and outcomes generated by DTCs. Clearly, due to the distributed nature of a DT, the shared storage is not constrained to a single physical repository but may encompass multiple storage systems tailored to specific requirements. The organization, storage, and retrieval of data to/from the SharedStorage component is managed by the StorageManager component, which exposes a *CRUDop* interface offering create, read, update, and delete operations. Functionalities of both DataProcessor and StorageManager are encapsulated within the DataManager component, which exposes the *CRUDop* interface provided by StorageManager directly to other DT components.

#### ShadowManager, ModelManager, ModelEngine, Simulator and TwinManager.

ShadowManager component directly maps onto the respective entity of the MTV, and is responsible for managing shadow instances. Moreover, it maps onto DigitalShadow entity and, in part, onto DigitalRepresentation entity, from which both DigitalShadow and DigitalModel derive. As mentioned before, it accesses the data provided by the DataProcessor through a port attachment relationship. Alternatively, depending on software requirements, it may directly access the shared storage through the *CRUDop* interface. As illustrated in Figure 5, ShadowManager provides the `getShadow` API, depicted using UML lollipop notation, enabling queries on shadows based on attributes such as type, timestamp, or name.

ModelManager component directly corresponds to the respective entity in the MTV and, consequently, to Digital Representation entity from which a model is derived. It is responsible for managing the lifecycle of digital models, including their creation, update, and configuration, ensuring consistency and accuracy across the models. ModelEngine component handles the execution of simulations, generating results based on modeled scenarios and providing insights into potential system behaviors under various conditions. It provides the *modelExecution* interface, which is utilized by ModelManager to execute digital models.

Together, they constitute Simulator component, which combines the virtualization of real-world systems through digital models with their execution in different scenarios. Moreover, Simulator offers the *getSimState* and *scenarioSim* APIs for simulating scenarios, enabling actions on digital models, adjusting simulation parameters, and retrieving the current simulation state. These functionalities are utilized by TwinManager (that directly maps onto the respective entity of the MTV) for comprehensive system orchestration to coordinate the interactions with other service-related components in order to facilitate seamless integration and efficient execution of Digital Twin services.

### StateMonitor.

StateMonitor component enables to concretely realize the monitoring capabilities of a DT and can thus be mapped onto ServiceManager entity. In particular, it offers the *getState* interface to TwinManager to retrieve the current state of the Physical Twin. The StateMonitor component, in particular, is responsible for collecting both real-world and simulated states, calculating the current state of the Physical Twin, storing in the shared storage for further analysis, and potentially presenting it to users via human-machine interfaces.

StateMonitor is fully supported by Azure Digital Twins and FIWARE. Azure Stream Analytics handles state data collection and analysis, while FIWARE leverages Cosmos for big data analysis. Eclipse Ditto provides partial support with event-handling mechanisms for processing state changes.

**Exemplars of component usage in a real scenario:** In the context of urban mobility and traffic flow management, StateMonitor continuously collects data from traffic sensors, such as vehicle counts at intersections, average speeds on roads, and real-time GPS data from public transports. These data are combined with simulated traffic states generated by Simulator. For instance, StateMonitor may calculate the current state as: “Intersection A has a traffic density of 80% with an average vehicle speed of 15 km/h”.

TwinManager accesses the traffic state through the *getState* interface to coordinate actions. For instance, based on the retrieved traffic data, TwinManager might initiate a prediction of alternative traffic flow scenarios to optimize road usage and alleviate congestion.

### DeviationDetector, Predictor, and Analyzer.

DeviationDetector, Predictor and Analyzer components specialize ServiceManager entity for what concerns the identification of anomalies or deviations, by comparing real or predicted states with expected states. This functionality allows the system to promptly detect discrepancies. In particular,

Predictor forecasts future system states using current and historical data, supporting proactive decision-making.

DeviationDetector identifies deviations based on the predicted state and the current state. Analyzer integrates the functionalities of DeviationDetector and Predictor, enabling analysis of real and simulated states and exposing the *prediction* API offered by the Predictor component to TwinManager, which uses it to trigger forecasting mechanisms.

DeviationDetector, Predictor, and Analyzer are fully supported by Azure Digital Twins, partially by FIWARE, and minimally by Ditto. Azure DTs provides robust tools like the Multivariate Detection Toolkit, Azure Stream Analytics, and Azure Machine Learning for detection, prediction, and analysis. FIWARE supports these functionalities with Cosmos and Perseo for rule-based detection and forecasting. In contrast, Ditto offers limited data analytics capabilities and lacks native support for deviation detection or prediction.

### ScenarioGenerator, SolutionFinder, and Planner.

ScenarioGenerator, SolutionFinder, and Planner components specialize ServiceManager entity for what concerns the generation of diverse scenarios to simulate and evaluate the system’s behavior under varying conditions, facilitating the exploration of alternative strategies. In particular, SolutionFinder determines the optimal set of actions required to restore the system to its desired state when deviations or anomalies are identified. Building on this, Planner leverages the solutions identified by SolutionFinder and analyzes the simulation results from scenarios created by ScenarioGenerator. This enables the Planner to devise and execute effective corrective actions, ensuring the system’s resilience and stability.

As shown in Fig. 5, SolutionFinder is connected to DeviationDetector through a port attachment relationship to receive real or predicted deviations. Additionally, ScenarioGenerator offers the *genScenario* interface used by SolutionFinder. Finally, Planner interacts with TwinManager through the *newScenarioSim* API. This allows the Planner to simulate alternative scenarios by leveraging the TwinManager’s access to underlying simulators.

SolutionFinder, ScenarioGenerator, and Planner lack direct support from the selected technologies. Azure Digital Twins, Eclipse Ditto, and FIWARE do not provide dedicated tools for scenario generation, solution finding, or planning functionalities. These components would require custom implementations or integrations with external simulation and planning frameworks to achieve their intended purpose.

### FeedbackExecutor.

FeedbackExecutor component maps onto the FeedbackProvider entity of the MTV and plays a dual role: when DeviationDetector identifies a deviation but no corrective plan is provided by the DT, FeedbackExecutor generates alerts or warnings for the physical system. Conversely, when Planner provides a plan, FeedbackExecutor translates this plan into actionable instructions executable by the physical system.

**Table 4**

Traceability Twin View: Matrix Diagram.

DTE/DTC Name	Physical Twin	Data Provider	Data Receiver	Adapter	P2D Adapter	D2P Adapter	Digital Repres.	Digital Shadow	Shadow Manager	Digital Model	Model Manager	Twin Manager	Service Manager	Feedback Provider	Data Manager	Data Model
PhysicalTwin	✓															
DataProvider		✓														
DataReceiver			✓													
P2DAdapter				✓	✓											
D2PAdapter					✓	✓										
DataProcessor															✓	✓
StorageManager															✓	
DataManager															✓	
SharedStorage															✓	✓
ShadowManager								✓	✓	✓						
ModelManager											✓					
ModelEngine												✓				
Simulator									✓	✓	✓					
TwinManager													✓			
StateMonitor														✓		
DeviationDet.														✓		
Predictor														✓		
Analyzer														✓		
SolutionFinder														✓		
ScenarioGen.														✓		
Planner														✓		
FeedbackExec.															✓	

Then, as anticipated, D2PAdapter ensures that these instructions are transformed into a format compatible with the physical infrastructure.

FeedbackExecutor is not natively supported by the selected technologies, as its implementation depends on custom solutions tailored to the physical system and the specific objectives of the DT.

**Exemplars of component usage in a real scenario:** If a traffic congestion is detected on Main Street without a plan, the FeedbackExecutor may generate an alert: *“High congestion detected on Main Street; notify drivers to avoid the area”*. When a plan is available, such as *“Divert vehicles from Main Street to Elm Avenue and extend green light duration on Elm Avenue by 20 seconds”*, the FeedbackExecutor translates it into commands for traffic control systems, e.g traffic lights and dynamic road signs.

#### 4.4. Traceability Twin View

The *Traceability Twin View* serves as the bridge between the module and component views of the proposed TwinArch. As outlined by the SEI [30], the architectural elements across different views must be interrelated to maintain coherence across abstraction levels, ensuring that high-level domain concepts are effectively aligned with the software components that implement them.

The separation of concerns inherent to this approach is pivotal. The Module Twin View focuses on defining the system’s objectives and functionalities, outlining what the system is designed to accomplish. In contrast, the Component Twin View delves into the structural and operational aspects, detailing how these objectives are realized. By connecting these two perspectives, the Traceability Twin View ensures consistency, enhances clarity, and fosters alignment across all levels of the architecture, providing a unified understanding of the system’s design and implementation.

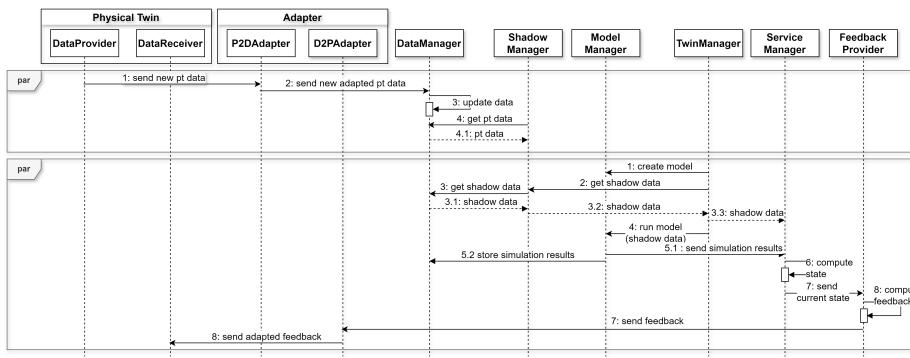
The *Matrix Diagram* in Table 4 illustrates the relationships between MTV entities and their corresponding CTV components. As discussed in the previous subsections, some mappings are straightforward, such as PhysicalTwin, DataProvider, and DataReceiver. These elements primarily relate to the physical system, and their further refinement is domain-dependent. In other cases, domain entities are

mapped to multiple components. For instance, DataManager and DataModel entities correspond to DataProcessor, Storage Manager, and SharedStorage components. This mapping is driven by the adoption of the shared-data architectural style in CTV, designed to handle the foundational role of data in a Digital Twin system.

While mappings for digital shadows and models are clear, ServiceManager exemplifies a more nuanced case. This class abstracts the idea of services in the module view, while in the CTV the concept of DT services is refined through the introduction of the following components: StateMonitor, DeviationDetector, Predictor, Analyzer, SolutionFinder, ScenarioGenerator, and Planner. These components reflect the Digital Twin system’s service-oriented capabilities, supporting monitoring, deviation detection, prediction, data analysis, and planning.

#### 4.5. Dynamic Twin View

The *Dynamic Twin View* captures the runtime interactions between the architectural elements of the DT to achieve specific functionalities. This perspective is illustrated through two use cases. The first use case focuses on *monitoring*, which entails the continuous tracking and analysis of the physical system’s state by combining simulations of digital models with real-world data. The second use case addresses the *prediction*, leveraging the DT’s simulation and analytical capabilities to forecast future states or behaviors of the physical system, including identifying potential issues, evaluating alternative scenarios, and recommending corrective actions. Both use cases are represented using *UML Sequence Diagrams*. The monitoring use case, which employs DT domain entities, is detailed in Section 4.5.1, while the prediction use case, utilizing DT components, is explained in Section 4.5.2. This distinction emphasizes the modeling of dynamic interactions at varying levels of abstraction (entities and components) to effectively address diverse functionalities.



**Figure 6:** Dynamic Twin View: UML Sequence Diagram of Monitoring Use Case with domain entities.

#### 4.5.1. Use Case: Monitoring Service

Figure 6 illustrates the *UML Sequence Diagram* of monitoring use case. The process begins with *DataProvider* transmitting data collected from the physical twin to the DT system. These data are processed by *P2DAdapter*, which converts it into a format compatible with the DT. The adapted data are subsequently forwarded to the *DataManager* to efficiently store and manage them. *ShadowManager* retrieves these data from *DataManager* (or directly via *P2DAdapter* to reduce latency) to update the relevant digital shadows. These shadows represent real-world states grouped by shadow types (e.g., traffic packets or other domain-specific data types).

As *PhysicalTwin* continues to send data, *ModelManager* is engaged to create or update digital models, used by *TwinManager* to execute simulations by feeding the models with the real-world data. The results of these simulations are stored in the system, providing enhanced insights into system behavior. In the context of monitoring, *ServiceManager* computes the state of the physical system by combining the simulation results with the shadow data. This computed state is then delivered to *FeedbackProvider*, which generates feedback messages. For instance, a simple message such as “*system ok*” may be generated, completing the monitoring loop. This feedback enables actionable responses, allowing the physical system to adjust as needed.

#### 4.5.2. Use Case: Prediction Service

Figure 7 illustrates the *UML Sequence Diagram* for the prediction use case, in which the DT forecasts future states of the physical system, identifies potential failures, and determines optimal actions to prevent them. The process begins with *TwinManager* initiating a prediction request, which prompts *Predictor* component to forecast future states of the system.

*Predictor* transmits the computed future states to the *DeviationDetector*, responsible for identifying potential deviations by comparing the predicted states against predefined thresholds or expected values. If a deviation is detected, *DeviationDetector* notifies *SolutionFinder* to initiate the deviation-handling process. *SolutionFinder* collaborates with *ScenarioGenerator* to explore and evaluate alternative scenarios for resolving the detected deviation.

When *SolutionFinder* identifies the possible solutions to address the problem, it triggers *ScenarioGenerator* to create detailed simulations of these solutions. Therefore, the execution of these new scenarios are triggered by interacting with *Simulator* through *TwinManager*. The manager facilitates the communication to update or create digital models via *ModelManager*, which subsequently triggers models execution by invoking *ModelEngine*.

These simulations provide critical insights into potential outcomes, enabling *SolutionFinder* to compute the optimal solution plan. Once the solution plan is finalized, it is transmitted to *FeedbackExecutor*, which ensures that the plan is translated into actionable feedback for *PhysicalTwin*. The feedback is formatted appropriately using *D2PAdapter* and sent to *DataReceiver*. Finally, the physical system applies this feedback to adjust its operations, thereby closing the prediction loop.

## 5. Online Survey

This Section presents the outcomes of the validation conducted with industry and academic experts through the online questionnaire<sup>11</sup>. Detailed results are available in the replication package.

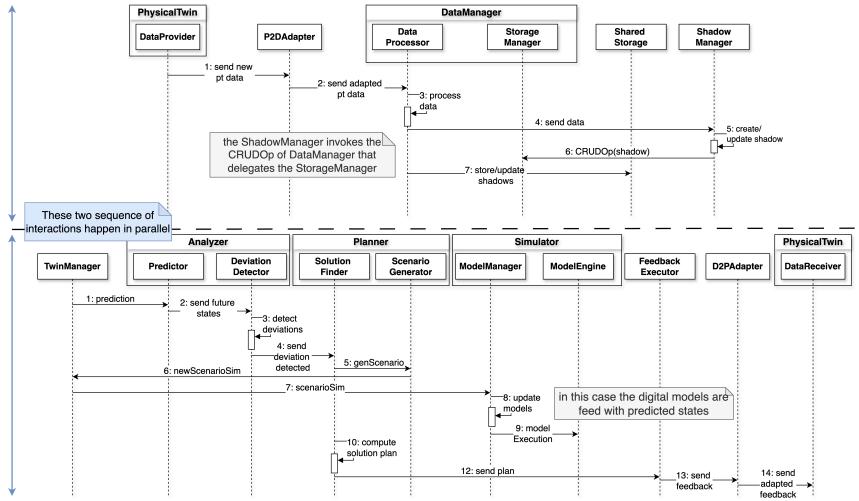
The final validation aims to evaluate whether the proposed TwinArch is complete, useful, and usable, ensuring the reference architecture’s effectiveness in facilitating the design and implementation of DT systems across diverse domains. To achieve this goal, three hypotheses were formulated for each architectural view:

- H1.* The Architectural View adequately represents all necessary elements (*completeness*).
- H2.* The Architectural View provides practical value for DT design and development (*usefulness*).
- H3.* The Architectural View is intuitive and easy to apply in practice (*perceived usability*).

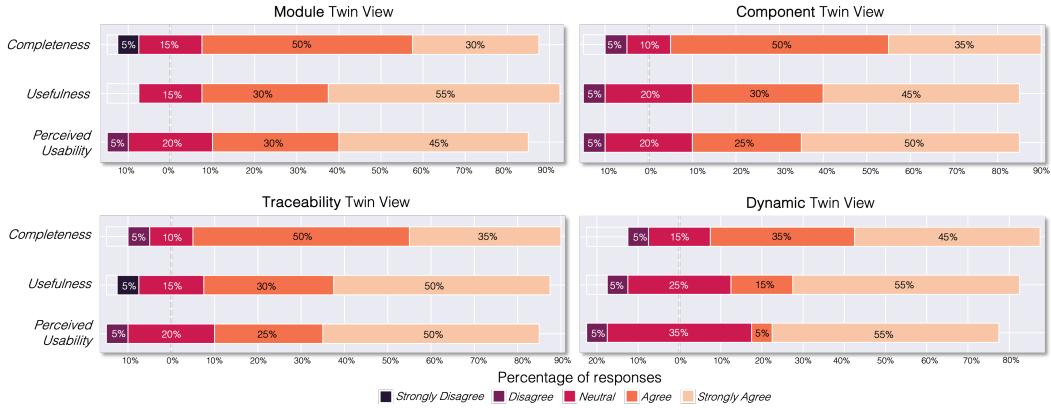
These attributes are crucial for capturing practitioners’ evaluations of the core dimensions that determine the impact of the proposed architecture [81, 82]. **Completeness** ensures that the architectural view includes all essential elements,

<sup>11</sup> Available at: <https://alessandrasomma28.github.io/twinarch/validation.html>

## TwinArch: A Digital Twin Reference Architecture



**Figure 7:** Dynamic Twin View: UML Sequence Diagram of Prediction Use Case with components.



**Figure 8:** Likert plot for questionnaire answers.

avoiding any gaps that could compromise system functionality. **Usefulness** confirms that the architecture delivers practical value by effectively supporting and guiding the design and development of Digital Twin systems. Lastly, **perceived usability** represents how easy to use the given artifacts are perceived by the practitioners.

### 5.1. Questionnaire results

Figure 8 presents a summary of the questionnaire responses, capturing the perceptions of all 20 respondents regarding the completeness, usefulness, and perceived usability of TwinArch.

As shown in the Likert plots, TwinArch is widely considered complete, with a significant majority affirming the completeness of its module, component, traceability, and dynamic views (35–50% agree, 30–45% strongly agree with hypothesis H1). Many respondents emphasized that TwinArch “aims at giving a very complete architectural description of how a Digital Twin should be structured” and considered it particularly important because “at the moment a unified definition/description of a Digital Twin system is not present”. The 15% of respondents provided a neutral response, which,

as justified in open-ended questions, stemmed from the perception that completeness is subjective and dependent on the specific goals of the Digital Twin being developed. However, a small percentage (5%) disagreed or strongly disagreed with the completeness of TwinArch, citing the lack of architectural elements for user interfaces responsible for external interfacing and enhancement of data management aspects.

TwinArch is positively evaluated in terms of usefulness, with 45–55% of participant strongly agreeing with hypothesis H2. Practitioners highlighted that it is “*a well-established way to document architecture and design a DT*”, particularly facilitating discussions among diverse groups such as stakeholders, system engineers, and developers. The selection of the UML language received positive feedback, and the traceability between views was also appreciated for its ability to map elements effectively, with one respondent stating that “*there is coherence among the various views*”. Respondents acknowledged that TwinArch “*gives some useful guidelines that can be followed by developers and researchers in the Digital Twin domain*” and provides “*a clear and practical reference guideline, making it valuable for researchers*,

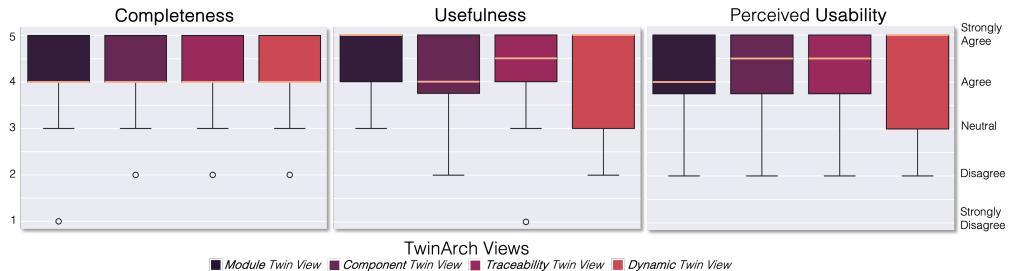


Figure 9: Box plots for questionnaire answers.

*companies, and developers aiming to implement digital twin solutions*". These comments emphasize the practical usefulness of TwinArch as a guideline for developing Digital Twins across various domains.

Regarding perceived usability, the responses included more neutral and disagreeing opinions compared to the other attributes. While practitioners recognized that TwinArch "can be part of a DT standard", the ease of use aspect generated some disagreement. This was primarily attributed to the domain independence of the proposal, as respondents highlighted that "*it would have to be declined in the future on the specific applications to make it operational*". Some practitioners noted the "*lack of practical examples*" and "*limited stakeholder engagement*" as factors hindering usability. Additionally, the ease of use was perceived as challenging when trying to extend TwinArch adoption to stakeholders with "*limited technical background or familiarity with software architecture*".

### 5.1.1. Statistical Analysis

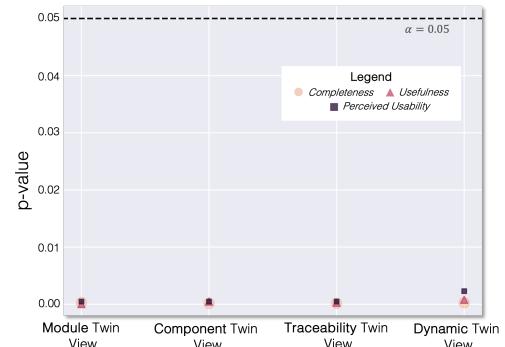
In addition to the Likert plot analysis, we performed a detailed statistical analysis of the responses. First, we used boxplots to visualize where the hypotheses  $H1$ ,  $H2$ , and  $H3$  hold for each view. Considering that responses follow a Likert scale where Strongly Disagree = 1, Disagree = 2, Neutral = 3, Agree = 4, and Strongly Agree = 5, the resulting boxplots are shown in Figure 9. These plots compare the three attributes—completeness, usefulness, and perceived usability—across the different views. As rendered in Figure 9, the medians for all attributes consistently fall within the Agree and Strongly Agree categories across all views, reflecting an overall positive perception of TwinArch. This indicates that respondents perceive the synthesized TwinArch as complete, useful, and usable, consistent with the findings from the Likert plot analysis.

Additionally, we conducted a deeper statistical analysis to examine the completeness, usefulness, and perceived usability of the proposed reference architecture. To achieve this, we transformed the responses as follows: Strongly Disagree and Disagree (1 and 2) were mapped to -1, Neutral (3) to 0, and Agree and Strongly Agree (4 and 5) to 1. This transformation creates a symmetric scale centered at zero, ensuring that negative and positive responses are equally distanced. By summing the transformed responses for each respondent, we determine whether the aggregate

score supports the hypotheses ( $\sum > 0$ ) or contradicts them ( $\sum < 0$ ).

To determine the appropriate statistical test, we first assessed data normality using the Shapiro-Wilk test, which indicated that data did not follow a normal distribution. Consequently, we employed the non-parametric one-sample Wilcoxon test to evaluate statistical and practical significance.

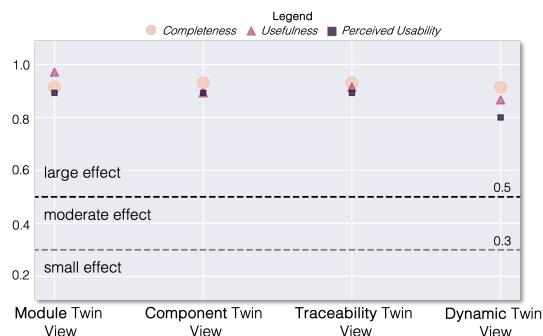
**Statistical Significance.** We tested our hypotheses for statistical significance using the one-sample Wilcoxon signed-rank test, which determines whether the aggregate responses for each view and dimension are significantly greater than 0 at a 5% significance level. The results are illustrated in Figure 10.

Figure 10: Wilcoxon one-sample test for statistical significance (hypothesis:  $\mu \geq 0$ ).

We note that all p-values for each view and each dimension are below the  $\alpha = 0.05$  threshold, indicating statistically significant results. Additionally, the majority of p-values are very close to 0, confirming that most respondents rated the TwinArch attributes well above the neutral value. The results validate that the correlation between the TwinArch views and the quality attributes (completeness, usefulness, and perceived usability) is statistically significant. This finding further underscores the relevance and applicability of TwinArch as a reference architecture for DT systems.

**Practical Significance.** We tested our hypotheses for practical significance calculating the Cohen's measure of effect size  $r$  derived from the one-sample Wilcoxon signed-rank test. The test evaluates whether the effect size is greater than zero for the sum of all respondents' answers across

combinations of views and attributes. The resulting plot is shown in Figure 11.



**Figure 11:** Effect size for one sample Wilcoxon test for practical significance.

According to Cohen J. [83], the thresholds for effect size are defined as small effect  $< 0.3$ , moderate effect  $> 0.3$  and  $< 0.5$ , and large effect  $> 0.5$ . From Fig. 11, it is clear that all effect sizes exceed the large effect's threshold across all views and attributes. This demonstrates the practical significance of TwinArch's completeness, usefulness, and perceived usability.

More in detail, completeness and usefulness consistently show higher effect sizes across all views compared to perceived usability, suggesting that participants found the former attributes to be particularly impactful. This difference may be attributed to the limited availability of practical Digital Twin examples, which would allow for a more thorough evaluation of the ease-of-use attribute. Nevertheless, while slightly lower, perceived usability still falls well within the range of a large effect. These findings confirm that TwinArch not only achieves statistical significance but also demonstrates meaningful practical utility across its views and attributes.

## 6. Discussion

This Section discusses the limitations identified in the selected studies on Digital Twin architectures, and how TwinArch addresses these challenges. Additionally, we examine the mapping between TwinArch's elements and the ISO 23247 functional entities to assess its alignment with existing relevant standards, identifying both gaps and enhancements introduced by TwinArch.

**Lack of multi-view documentation.** As already pointed out in Sections 1 and 2, a major limitation in existing Digital Twin architectures is the lack of structured multi-view documentation [29, 55]. Many architectures rely on single-diagram representations that combine different abstraction levels [23, 52], leading to confusion and misinterpretation of architectural elements. Additionally, unclear relationships between elements and the mixing of structural and dynamic aspects make it difficult to understand the runtime behavior of DT systems in use cases such as monitoring or prediction [37, 38, 67, 71].

TwinArch addresses these issues by adopting the SEI Views and Beyond method, resulting in a structured, multi-view architecture. By clearly delineating distinct architectural views, TwinArch prevents the incorrect assignment of elements to inappropriate levels of abstraction. Additionally, traceability between views enhances clarity, allowing stakeholders to seamlessly navigate between high-level conceptual overviews and detailed technical specifications. The introduction of a dedicated view for dynamic aspects and the modeling of different use cases in separate diagrams significantly improves the understanding of DT runtime behavior.

**Lack of data-related aspects.** A key limitation in existing DT architectures is the overemphasis on simulation functionalities, while data-related aspects remain underdeveloped despite their central role in DT systems. Many proposals focus primarily on virtual modeling [21, 28, 68, 69], often neglecting critical aspects such as data management, bidirectional data exchange, and data adaptation.

TwinArch addresses this gap by adopting a data-centric architectural approach, utilizing a shared-data style to decouple data producers from consumers through a shared repository for data exchange. This enhances modifiability and scalability, facilitating efficient data handling across various DT applications. Additionally, TwinArch explicitly integrates data adaptation and shadowing as core architectural components, ensuring these essential aspects are systematically incorporated rather than treated as secondary considerations.

**Domain dependence of DT architectures.** Another challenge limiting the widespread adoption of DTs is the strong domain dependence of state-of-the-art architectures. Many existing solutions are tailored to specific industries, such as manufacturing, aerospace, and healthcare, making them less adaptable to cross-domain applications [19, 28, 53, 57].

TwinArch overcomes this limitation by introducing a domain-independent reference architecture, designed for wide applicability across various DT domains. Its architectural elements are mapped onto multiple DT development platforms without being tied to a specific application domain. This allows practitioners to leverage reusable and concrete artifacts, which can be customized to meet domain-specific requirements.

**Gaps in the ISO 23247 standard.** As noted in [18], alongside its domain dependence, the ISO 23247 standard lacks comprehensive coverage of key architectural elements, particularly those related to data management. Its reliance on an entity-based reference model, with limited attention to runtime behaviors and interactions, and the absence of concrete artifacts to support practitioners, poses significant challenges for the effective development and implementation of Digital Twin solutions.

TwinArch addresses these shortcomings while aligning its architectural elements with the ISO 23247 Functional Entities. Table 5 presents the mapping between the standard's FEs, their respective domains as defined in ISO 23247, and the corresponding Module Twin View and Component

**Table 5**

TwinArch's elements mapping to ISO 23247 Functional Entities.

ISO 23247 Functional Entity	ISO 23247 Domain	Module Twin View Element	Component Twin View Element
Observable Manufacturing Elements	Observable Manufacturing Domain	PhysicalTwin	PhysicalTwin
Data Collecting	Data Collection and Device Control Domain	DataProvider	DataProvider
Data Pre-Processing	Data Collection and Device Control Domain	DataManager	DataProcessor
Data Translation	Cross-System Domain	Adapter, P2DAdapter, D2PAdapter	P2DAdapter, D2PAdapter
Controlling	Data Collection and Device Control Domain	FeedbackProvider	FeedbackExecutor
Actuation	Data Collection and Device Control Domain	DataReceiver	DataReceiver
Digital Modeling	Core Domain	DigitalRepresentation, DigitalModel	ModelEngine
Maintenance	Core Domain	X	X
Synchronization	Core Domain	DataProvider, DataReceiver, TwinManager	DataProvider, DataReceiver, TwinManager
Simulation	Core Domain	ModelManager	Simulator
Analytic Service	Core Domain	ServiceManager	Analyzer
Reporting	Core Domain	TwinManager	TwinManager, StateMonitor
Application Support	Core Domain	TwinManager	TwinManager
Interoperability Support	Core Domain	DataModel, DataManager	DataProcessor, SharedStorage
Access Control	Core Domain	X	X
Security Support	Cross-System Domain	X	X
User Interface	User Domain	TwinManager	TwinManager

Twin View elements in TwinArch. This mapping highlights areas where TwinArch ensures compliance, providing a one-to-one alignment with key entities such as PhysicalTwin, DataProvider, and DigitalModel, thereby supporting the standard's foundational definitions.

However, the table also underscores critical omissions in the ISO 23247 standard. For instance, the standard does not address data-related entities, such as digital shadowing elements which play a pivotal role in ensuring historical state management and real-time synchronization. Similarly, service-related entities like those supporting planning, scenario generation, and prediction are absent in the standard, even though these functionalities are essential for DT implementations. While basic concepts such as simulation, monitoring, and analysis are covered, the lack of advanced service-oriented elements limits the standard's applicability for more complex DT use cases.

On the other hand, it is evident that TwinArch does not address non-functional requirements, such as security, access control, and maintenance, as these aspects fall outside the scope of the proposed architecture. By bridging the gaps in ISO 23247 and extending its capabilities, TwinArch provides a concrete and extended architecture for supporting practitioners in the design and development of DT solutions.

## 7. Conclusion and Future Work

This paper proposed a domain-independent and multi-view Digital Twin Reference Architecture, called TwinArch. TwinArch helps filling a research gap in the field since existing reference architectures and standards are tailored to a specific domain, like manufacturing, and are typically documented with a single view mixing static and dynamic aspects.

TwinArch has been built by carefully analyzing the state of the art in the field, as well as the most common DT development platforms. The proposed reference architecture has been also validated thanks to the help of 20 experts to check its completeness, usefulness, and perceived usability. We believe that TwinArch provides practitioners with practical artifacts that can be used to design and develop new DT

systems across various domains and to document existing ones.

As future work, we plan to put in practice TwinArch in various domains to better experiment its usefulness and perceived usability. We also believe that this study can help standard bodies to define a reference architecture standard.

## 8. Funding

This work has been partially supported by the Spoke 9 “Digital Society & Smart Cities” of ICSC - Centro Nazionale di Ricerca in High Performance-Computing, Big Data and Quantum Computing, funded by the European Union - NextGenerationEU (PNRR-HPC, CUP: E63C22000980007). Moreover, this work has been partially funded by: (a) the National Science Foundation under Grant No. 2232721; (b) the MUR (Italy) Department of Excellence 2023 - 2027; (c) the European HORIZON-KDT-JU research project MATISSE “Model-based engineering of Digital Twins for early verification and validation of Industrial Systems”, HORIZON-KDT-JU-2023-2-RIA, Prop. n.: 101140216-2, KDT232RIA\_00017; (d) the PRIN project P2022RSW5W - RoboChor: Robot Choreography; (e) the PRIN project 2022JKA4SL - HALO: etHical-aware AdjustabLe autOnomous systems.

## Data Availability

The SLR process, the analysis of the selected papers, the TwinArch documentation presented through UML diagrams, the high-quality figures, as well as the online survey questionnaire and its results, are accessible at: <https://alessandrasomma28.github.io/twinarch/>.

## References

- [1] D. Jones, C. Snider, A. Nassehi, J. Yon, B. Hicks, Characterising the digital twin: A systematic literature review, *CIRP Journal of Manufacturing Science and Technology* 29 (2020) 36–52. doi:10.1016/j.cirpj.2020.02.002.
- [2] H. van der Valk, H. Haße, F. Möller, B. Otto, Archetypes of digital twins, *Business & Information Systems Engineering* 64 (3) (2022) 375–391. doi:10.1007/s12599-021-00727-7.

- [3] F. Tao, H. Zhang, A. Liu, A. Y. C. Nee, Digital twin in industry: State-of-the-art, *IEEE Transactions on Industrial Informatics* 15 (4) (2019) 2405–2415. doi:[10.1109/TII.2018.2873186](https://doi.org/10.1109/TII.2018.2873186).
- [4] B. R. Barricelli, E. Casiraghi, D. Fogli, A survey on digital twin: Definitions, characteristics, applications, and design implications, *IEEE Access* 7 (2019) 167653–167671. doi:[10.1109/ACCESS.2019.2953499](https://doi.org/10.1109/ACCESS.2019.2953499).
- [5] S. R. Jeremiah, A. El Azzaoui, N. N. Xiong, J. H. Park, A comprehensive survey of digital twins: Applications, technologies and security challenges, *Journal of Systems Architecture* 151 (2024) 103120. doi:[10.1016/j.sysarc.2024.103120](https://doi.org/10.1016/j.sysarc.2024.103120).
- [6] R. Minerva, G. M. Lee, N. Crespi, Digital twin in the iot context: A survey on technical features, scenarios, and architectural models, *Proceedings of the IEEE* 108 (10) (2020) 1785–1824. doi:[10.1109/JPROC.2020.2998530](https://doi.org/10.1109/JPROC.2020.2998530).
- [7] Q. Qi, F. Tao, T. Hu, N. Anwer, A. Liu, Y. Wei, L. Wang, A. Y. Nee, Enabling technologies and tools for digital twin, *Journal of Manufacturing Systems* 58 (2021) 3–21.
- [8] T. Brockhoff, M. Heithoff, I. Koren, J. Michael, J. Pfeiffer, B. Rumpe, M. S. Uysal, W. M. P. Van Der Aalst, A. Wortmann, Process prediction with digital twins, in: 2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2021, pp. 182–187. doi:[10.1109/MODELS-C53483.2021.00032](https://doi.org/10.1109/MODELS-C53483.2021.00032).
- [9] Y. Wang, Z. Su, S. Guo, M. Dai, T. H. Luan, Y. Liu, A survey on digital twins: Architecture, enabling technologies, security and privacy, and future prospects, *IEEE Internet of Things Journal* 10 (17) (2023) 14965–14987. doi:[10.1109/JIOT.2023.3263909](https://doi.org/10.1109/JIOT.2023.3263909).
- [10] I. Abdullahi, S. Longo, M. Samie, Towards a distributed digital twin framework for predictive maintenance in industrial internet of things (iiot), *Sensors* 24 (8) (2024). doi:[10.3390/s24082663](https://doi.org/10.3390/s24082663).
- [11] S. Mihai, M. Yaqoob, D. V. Hung, W. Davis, P. Towakel, M. Raza, M. Karamanoglu, B. Barn, D. Shetve, R. V. Prasad, H. Venkataraman, R. Trestian, H. X. Nguyen, Digital twins: A survey on enabling technologies, challenges, trends and future prospects, *IEEE Communications Surveys & Tutorials* 24 (4) (2022) 2255–2291. doi:[10.1109/COMST.2022.3208773](https://doi.org/10.1109/COMST.2022.3208773).
- [12] S. Gil, P. H. Mikkelsen, C. Gomes, P. G. Larsen, Survey on open-source digital twin frameworks—a case study approach, *Software: Practice and Experience* 54 (6) (2024) 929–960. doi:<https://doi.org/10.1002/spe.3305>.
- [13] L. Garcés, S. Martínez-Fernández, L. Oliveira, P. Valle, C. Ayala, X. Franch, E. Y. Nakagawa, Three decades of software reference architectures: A systematic mapping study, *Journal of Systems and Software* 179 (2021) 111004. doi:[10.1016/j.jss.2021.111004](https://doi.org/10.1016/j.jss.2021.111004).
- [14] Automation systems and integration — digital twin framework for manufacturing, Standard, ISO (10 2021).  
URL <https://www.iso.org/standard/75066.html>
- [15] A. Shtofenmakher, G. Shao, Adaptation of iso 23247 to aerospace digital twin applications-on-orbit collision avoidance and space-based debris detection, in: AIAA SCITECH 2024 Forum, National Institute of Standards and Technology, 2024. doi:[10.2514/6.2024-0275](https://doi.org/10.2514/6.2024-0275).
- [16] J. Cederbladh, E. Ferko, E. Lundin, Towards adopting a digital twin framework (iso 23247) for battery systems, in: S. Latifi (Ed.), ITNG 2024: 21st International Conference on Information Technology-New Generations, Springer Nature Switzerland, Cham, 2024, pp. 397–404.
- [17] M.-S. Kang, D.-H. Lee, M. S. Bajestani, D. B. Kim, S. D. Noh, Edge computing-based digital twin framework based on iso 23247 for enhancing data processing capabilities, *Machines* 13 (1) (2025). doi:[10.3390/machines13010019](https://doi.org/10.3390/machines13010019).
- [18] E. Ferko, A. Bucaioni, P. Pelliccione, M. Behnam, Standardisation in digital twin architectures in manufacturing, in: 2023 IEEE 20th International Conference on Software Architecture (ICSA), 2023, pp. 70–81. doi:[10.1109/ICSA56044.2023.00015](https://doi.org/10.1109/ICSA56044.2023.00015).
- [19] T. Bolender, G. Burvenich, M. Dalibor, B. Rumpe, A. Wortmann, Self-adaptive manufacturing with digital twins, in: 2021 International symposium on software engineering for adaptive and self-managing systems (SEAMS), IEEE, 2021, pp. 156–166. doi:[10.1109/SEAMS51251.2021.00029](https://doi.org/10.1109/SEAMS51251.2021.00029).
- [20] A. Macéas, E. Navarro, C. E. Cuesta, U. Zdun, Architecting digital twins using a domain-driven design-based approach\*, in: 2023 IEEE 20th International Conference on Software Architecture (ICSA), 2023, pp. 153–163. doi:[10.1109/ICSA56044.2023.00022](https://doi.org/10.1109/ICSA56044.2023.00022).
- [21] R. van Dinter, B. Tekinerdogan, C. Catal, Reference architecture for digital twin-based predictive maintenance systems, *Computers & Industrial Engineering* 177 (2023) 109099. doi:[10.1016/j.cie.2023.109099](https://doi.org/10.1016/j.cie.2023.109099).
- [22] A. De Benedictis, F. Flammini, N. Mazzocca, A. Somma, F. Vitale, Digital twins for anomaly detection in the industrial internet of things: Conceptual architecture and proof-of-concept, *IEEE Transactions on Industrial Informatics* 19 (12) (2023) 11553–11563. doi:[10.1109/TII.2023.3246983](https://doi.org/10.1109/TII.2023.3246983).
- [23] F. Tao, X. Sun, J. Cheng, Y. Zhu, W. Liu, Y. Wang, H. Xu, T. Hu, X. Liu, T. Liu, Z. Sun, J. Xu, J. Bao, F. Xiang, X. Jin, maketwin: A reference architecture for digital twin software platform, *Chinese Journal of Aeronautics* 37 (1) (2024) 1–18. doi:[10.1016/j.cja.2023.05.002](https://doi.org/10.1016/j.cja.2023.05.002).
- [24] International standard for software, systems and enterprise—architecture description, Standard, IEEE/ISO/IEC (11 2022). doi:[10.1109/IEEESTD.2022.9938446](https://doi.org/10.1109/IEEESTD.2022.9938446).
- [25] S. Malakuti, J. Schmitt, M. Platenius-Mohr, S. Grüner, R. Gitzel, P. Bihani, A four-layer architecture pattern for constructing and managing digital twins, in: Software Architecture, Springer International Publishing, Cham, 2019, pp. 231–246.
- [26] E. Ferko, A. Bucaioni, M. Behnam, Architecting digital twins, *IEEE Access* 10 (2022) 50335–50350. doi:[10.1109/ACCESS.2022.3172964](https://doi.org/10.1109/ACCESS.2022.3172964).
- [27] H. Boyes, T. Watson, Digital twins: An analysis framework and open issues, *Computers in Industry* 143 (2022) 103763. doi:[10.1016/j.compind.2022.103763](https://doi.org/10.1016/j.compind.2022.103763).
- [28] A. Redelinghuys, A. Basson, K. Kruger, A six-layer architecture for the digital twin: a manufacturing case study implementation, *Journal of Intelligent Manufacturing* 31 (08 2020). doi:[10.1007/s10845-019-01516-6](https://doi.org/10.1007/s10845-019-01516-6).
- [29] C. Steinmetz, G. N. Schroeder, R. N. Rodrigues, A. Rettberg, C. E. Pereira, Key-components for digital twin modeling with granularity: Use case car-as-a-service, *IEEE Transactions on Emerging Topics in Computing* 10 (1) (2022) 23–33. doi:[10.1109/TETC.2021.3131532](https://doi.org/10.1109/TETC.2021.3131532).
- [30] P. Clements, F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, P. Merson, R. Nord, J. Stafford, Documenting Software Architectures: Views and Beyond, SEI Series in Software Engineering, Addison-Wesley, 2010.
- [31] G. Shao, S. P. Frechette, V. Srinivasan, An analysis of the new iso 23247 series of standards on digital twin framework for manufacturing, 2023 MSEC Manufacturing Science & Engineering Conference, New Brunswick, NJ, US, 2023.  
URL [https://tsapps.nist.gov/publication/get\\_pdf.cfm?pub\\_id=935765](https://tsapps.nist.gov/publication/get_pdf.cfm?pub_id=935765)
- [32] D. Bong Kim, G. Shao, G. Jo, A digital twin implementation architecture for wire+arc additive manufacturing based on iso 23247, *Manufacturing Letters* 34 (2022) 1–5. doi:[10.1016/j.mfglet.2022.08.008](https://doi.org/10.1016/j.mfglet.2022.08.008).
- [33] P. Spaney, S. Becker, R. Ströbel, J. Fleischer, S. Zenhari, H.-C. Möhring, A.-K. Spelttstößer, A. Wortmann, A model-driven digital twin for manufacturing process adaptation, in: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2023, pp. 465–469. doi:[10.1109/MODELS-C59198.2023.00081](https://doi.org/10.1109/MODELS-C59198.2023.00081).
- [34] V. Melo, J. Barbosa, G. Mota, F. d. L. Prieta, P. Leitao, Design of an iso 23247 compliant digital twin for an automotive assembly line, in: 2024 IEEE 7th International Conference on Industrial Cyber-Physical Systems (ICPS), 2024, pp. 1–6. doi:[10.1109/ICPS59941.2024.10640052](https://doi.org/10.1109/ICPS59941.2024.10640052).
- [35] B. Wallner, B. Zwölfer, T. Trautner, F. Bleicher, Digital twin development and operation of a flexible manufacturing cell using iso 23247, *Procedia CIRP* 120 (2023) 1149–1154, 56th CIRP International Conference on Manufacturing Systems 2023. doi:[10.1016/j.procir.2023.09.140](https://doi.org/10.1016/j.procir.2023.09.140).

- [36] G. Caiza, R. Sanz, Immersive digital twin under iso 23247 applied to flexible manufacturing processes, *Applied Sciences* 14 (10) (2024). doi:10.3390/app14104204.
- [37] S. Cuñat Negueroles, R. Reinosa Simón, M. Julián, A. Belsa, I. Laclale, R. S-Julián, C. E. Palau, A blockchain-based digital twin for iot deployments in logistics and transportation, *Future Generation Computer Systems* 158 (2024) 73–88. doi:10.1016/j.future.2024.04.011.
- [38] C. Rodríguez-Alonso, I. Pena-Regueiro, O. García, Digital twin platform for water treatment plants using microservices architecture, *Sensors* 24 (5) (2024). doi:10.3390/s24051568.
- [39] R. Wieringa, Design Science Methodology for Information Systems and Software Engineering, Springer, 2014. doi:10.1007/978-3-662-43839-8.
- [40] R. Wieringa, Design science as nested problem solving, in: Proceedings of the 4th International Conference on Design Science Research in Information Systems and Technology, DESRIST '09, Association for Computing Machinery, 2009. doi:10.1145/1555619.1555630.
- [41] K. Petersen, S. Vakkalanka, L. Kuzniarz, Guidelines for conducting systematic mapping studies in software engineering: An update, *Information and Software Technology* 64 (2015) 1–18. doi:10.1016/j.infsof.2015.03.007.
- [42] B. Kitchenham, O. P. Brereton, D. Budgen, M. Turner, J. Bailey, S. Linkman, Systematic literature reviews in software engineering—a systematic literature review, *Information and software technology* 51 (1) (2009) 7–15.
- [43] P. Kruchten, The 4+1 view model of architecture, *IEEE Software* 12 (1995) 45–50. doi:10.1109/52.469759.
- [44] P. Clements, Comparing the sei's views and beyond approach for documenting software architecture with ansi-ieee 1471-2000 (2005) 30.
- [45] J. Pfeiffer, D. Lehner, A. Wortmann, M. Wimmer, Modeling capabilities of digital twin platforms - old wine in new bottles?, *Journal of Object Technology* 21 (3) (2022) 3:1–14, the 18th European Conference on Modelling Foundations and Applications (ECMFA 2022). doi:10.5381/jot.2022.21.3.a10.
- [46] D. Lehner, S. Gil, P. H. Mikkelsen, P. G. Larsen, M. Wimmer, An architectural extension for digital twin platforms to leverage behavioral modelsbehaviors, in: 2023 IEEE 19th International Conference on Automation Science and Engineering (CASE), 2023, pp. 1–8. doi:10.1109/CASE56687.2023.10260417.
- [47] D. Lehner, J. Pfeiffer, E.-F. Tinsel, M. M. Strljic, S. Sint, M. Vierhauser, A. Wortmann, M. Wimmer, Digital twin platforms: Requirements, capabilities, and future prospects, *IEEE Software* 39 (2) (2022) 53–61. doi:10.1109/MS.2021.3133795.
- [48] R. Martínez, J. A. Pastor, B. Álvarez, A. Iborra, A testbed to evaluate the fiware-based iot platform in the domain of precision agriculture, *Sensors* 16 (11) (2016).
- [49] D. S. Cruzes, T. Dyba, Recommended steps for thematic synthesis in software engineering, in: 2011 International Symposium on Empirical Software Engineering and Measurement, 2011, pp. 275–284. doi:10.1109/ESEM.2011.36.
- [50] M. Mahmoud, C. Semeraro, M. A. Abdelkareem, A. G. Olabi, Designing and prototyping the architecture of a digital twin for wind turbine, *International Journal of Thermofluids* 22 (2024) 100622. doi:10.1016/j.ijft.2024.100622.
- [51] A. Somma, A. De Benedictis, C. Esposito, N. Mazzocca, The convergence of digital twins and distributed ledger technologies: A systematic literature review and an architectural proposal, *Journal of Network and Computer Applications* 225 (2024) 103857. doi:10.1016/j.jnca.2024.103857.
- [52] Y. Wenqiang, X. Bao, Y. Zheng, L. Zhang, Z. Zhang, Z. Zhang, L. Li, A digital twin framework for large comprehensive ports and a case study of qingdao port, *The International Journal of Advanced Manufacturing Technology* 131 (12 2022). doi:10.1007/s00170-022-10625-1.
- [53] A. De Benedictis, N. Mazzocca, A. Somma, C. Strigaro, Digital twins in healthcare: An architectural proposal and its application in a social distancing case study, *IEEE Journal of Biomedical and Health Informatics* 27 (10) (2023) 5143–5154. doi:10.1109/JBHI.2022.3205506.
- [54] L. De Donato, R. Dirnfeld, A. Somma, A. De Benedictis, F. Flammini, S. Marrone, M. Saman Azari, V. Vittorini, Towards ai-assisted digital twins for smart railways: preliminary guideline and reference architecture, *Journal of Reliable Intelligent Environments* 9 (3) (2023) 303 – 317. doi:10.1007/s40860-023-00208-6.
- [55] H. Yu, D. Yu, C. Wang, Y. Hu, Y. Li, Edge intelligence-driven digital twin of cnc system: Architecture and deployment, *Robotics and Computer-Integrated Manufacturing* 79 (2023) 102418. doi:10.1016/j.rcim.2022.102418.
- [56] H. Elayan, M. Aloqaily, M. Guizani, Digital twin for intelligent context-aware iot healthcare systems, *IEEE Internet of Things Journal* 8 (23) (2021) 16749–16757. doi:10.1109/JIOT.2021.3051158.
- [57] J. A. Erkoyuncu, I. F. del Amo, D. Ariansyah, D. Bulka, R. Vrabič, R. Roy, A design framework for adaptive digital twins, *CIRP Annals* 69 (1) (2020) 145–148. doi:https://doi.org/10.1016/j.cirp.2020.04.086.
- [58] L. Adreani, P. Bellini, M. Fanfani, P. Nesi, G. Pantaleo, Smart city digital twin framework for real-time multi-data integration and wide public distribution, *IEEE Access* 12 (2024) 76277–76303. doi:10.1109/ACCESS.2024.3406795.
- [59] N. Hossein Motlagh, M. A. Zaidan, L. Lovén, P. L. Fung, T. Häneninen, R. Morabito, P. Nurmi, S. Tarkoma, Digital twins for smart spaces—beyond iot analytics, *IEEE Internet of Things Journal* 11 (1) (2024) 573–583. doi:10.1109/JIOT.2023.3287032.
- [60] J. Robles, C. Martín, M. Díaz, Opentwins: An open-source framework for the development of next-gen compositional digital twins, *Computers in Industry* 152 (2023) 104007. doi:10.1016/j.compind.2023.104007.
- [61] S. N. Gouriseti, S. Bhadra, D. J. Sebastian-Cardenas, M. Touhiduzzaman, O. Ahmed, A theoretical open architecture framework and technology stack for digital twins in energy sector applications, *Energies* 16 (13) (2023). doi:10.3390/en16134853.
- [62] J. Conde, A. Munoz-Arcentales, A. Alonso, G. Huecas, J. Salvachúa, Collaboration of digital twins through linked open data: Architecture with fiware as enabling technology, *IT Professional* 24 (6) (2022) 41–46. doi:10.1109/MITP.2022.3224826.
- [63] J. Conde, A. Munoz-Arcentales, A. Alonso, S. López-Pernas, J. Salvachúa, Modeling digital twin data and architecture: A building guide with fiware as enabling technology, *IEEE Internet Computing* 26 (3) (2022) 7–14. doi:10.1109/MIC.2021.3056923.
- [64] P. Eirinakis, S. Lounis, S. Plitsos, G. Arampatzis, K. Kalaboukas, K. Kenda, J. Lu, J. M. Rožanec, N. Stojanovic, Cognitive digital twins for resilience in production: A conceptual framework, *Information* 13 (1) (2022). doi:10.3390/info13010033.
- [65] B. Tekinerdogan, C. Verdouw, Systems architecture design pattern catalog for developing digital twins, *Sensors* 20 (18) (2020). doi:10.3390/s20185103.
- [66] M. Glatt, C. Sinnwell, L. Yi, S. Donohoe, B. Ravani, J. C. Aurich, Modeling and implementation of a digital twin of material flows based on physics simulation, *Journal of Manufacturing Systems* 58 (2021) 231–245, digital Twin towards Smart Manufacturing and Industry 4.0. doi:10.1016/j.jmsy.2020.04.015.
- [67] H. Sartaj, S. Ali, T. Yue, K. Moberg, Model-based digital twins of medicine dispensers for healthcare iot applications, *Software: Practice and Experience* 54 (6) (2024) 1172–1192. doi:10.1002/spe.3311.
- [68] G. N. Schroeder, C. Steinmetz, R. N. Rodrigues, R. V. B. Henriques, A. Rettberg, C. E. Pereira, A methodology for digital twin modeling and deployment for industry 4.0, *Proceedings of the IEEE* 109 (4) (2021) 556–567. doi:10.1109/JPROC.2020.3032444.
- [69] C. Su, Y. Han, X. Tang, Q. Jiang, T. Wang, Q. He, Knowledge-based digital twin system: Using a knowledge-driven approach for manufacturing process modeling, *Computers in Industry* 159–160 (2024) 104101. doi:10.1016/j.compind.2024.104101.
- [70] J. Pfeiffer, D. Lehner, A. Wortmann, M. Wimmer, Towards a product line architecture for digital twins, in: 2023 IEEE 20th International Conference on Software Architecture Companion (ICSA-C), 2023,

- pp. 187–190. doi:[10.1109/ICSA-C57050.2023.00049](https://doi.org/10.1109/ICSA-C57050.2023.00049).
- [71] T. Usländer, M. Baumann, S. Boschert, R. Rosen, O. Sauer, L. Stojanovic, J. C. Wehrstedt, Symbiotic evolution of digital twin systems and dataspaces, *Automation* 3 (3) (2022) 378–399. doi:[10.3390/automation3030020](https://doi.org/10.3390/automation3030020).
- [72] L. F. Rivera, M. Jiménez, N. M. Villegas, G. Tamura, H. A. Müller, The forging of autonomic and cooperating digital twins, *IEEE Internet Computing* 26 (5) (2022) 41–49. doi:[10.1109/MIC.2021.3051902](https://doi.org/10.1109/MIC.2021.3051902).
- [73] R. Eramo, F. Bordeleau, B. Combemale, M. v. d. Brand, M. Wimmer, A. Wortmann, Conceptualizing digital twins, *IEEE Software* 39 (2) (2022) 39–46. doi:[10.1109/MS.2021.3130755](https://doi.org/10.1109/MS.2021.3130755).
- [74] P. Spaney, S. Becker, R. Ströbel, J. Fleischer, S. Zenhari, H.-C. Möhring, A.-K. Splettstößer, A. Wortmann, A model-driven digital twin for manufacturing process adaptation, in: 2023 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C), 2023, pp. 465–469. doi:[10.1109/MODELS-C59198.2023.00081](https://doi.org/10.1109/MODELS-C59198.2023.00081).
- [75] N. Eskandani, S. Grüner, Cloud-native architecture for mixed file-based and api-based digital twin exchange, in: *Software Architecture*, Springer Nature Switzerland, Cham, 2023, pp. 292–299.
- [76] P. Bibow, M. Dalibor, C. Hopmann, B. Mainz, B. Rumpe, D. Schmalzing, M. Schmitz, A. Wortmann, *Model-Driven Development of a Digital Twin for Injection Molding*, Springer International Publishing, Cham, 2020, pp. 85–100. doi:[10.1007/978-3-030-49435-3\\_6](https://doi.org/10.1007/978-3-030-49435-3_6).
- [77] N. Mohamed, J. Al-Jaroodi, I. Jawhar, N. Kesserwan, Leveraging digital twins for healthcare systems engineering, *IEEE Access* 11 (2023) 69841–69853. doi:[10.1109/ACCESS.2023.3292119](https://doi.org/10.1109/ACCESS.2023.3292119).
- [78] A. Costantini, G. Di Modica, J. C. Ahouangonou, D. C. Duma, B. Martelli, M. Galletti, M. Antonacci, D. Nehls, P. Bellavista, C. Delamarre, D. Cesini, Iotwins: Toward implementation of distributed digital twins in industry 4.0 settings, *Computers* 11 (5) (2022). doi:[10.3390/computers11050067](https://doi.org/10.3390/computers11050067).
- [79] H. D. Perez, J. M. Wassick, I. E. Grossmann, A digital twin framework for online optimization of supply chain business processes, *Computers & Chemical Engineering* 166 (2022) 107972. doi:[10.1016/j.compchemeng.2022.107972](https://doi.org/10.1016/j.compchemeng.2022.107972).
- [80] R. Vrabič, J. A. Erkoyuncu, M. Farsi, D. Ariansyah, An intelligent agent-based architecture for resilient digital twins in manufacturing, *CIRP Annals* 70 (1) (2021) 349–352. doi:[10.1016/j.cirp.2021.04.049](https://doi.org/10.1016/j.cirp.2021.04.049).
- [81] R. Wohlrab, U. Eliasson, P. Pelliccione, R. Heldal, Improving the consistency and usefulness of architecture descriptions: Guidelines for architects, in: 2019 IEEE International Conference on Software Architecture (ICSA), 2019, pp. 151–160. doi:[10.1109/ICSA.2019.00024](https://doi.org/10.1109/ICSA.2019.00024).
- [82] L. Bass, B. E. John, Achieving usability through software architectural styles, in: CHI '00 Extended Abstracts on Human Factors in Computing Systems, CHI EA '00, Association for Computing Machinery, New York, NY, USA, 2000, p. 171–172. doi:[10.1145/633292.633387](https://doi.org/10.1145/633292.633387).
- [83] J. Cohen, A power primer, *Tutorials in Quantitative Methods for Psychology* 112 (07 1992). doi:[10.1037/0033-2909.112.1.155](https://doi.org/10.1037/0033-2909.112.1.155).