Architecting Digital Twins Using a Domain-Driven Design-Based Approach*

Aurora Macías† and Elena Navarro††
LoUISE Research Group
University of Castilla-La Mancha
Albacete, Spain, aurora.macias@alu.uclm.es
elena.navarro@uclm.es

Carlos E. Cuesta
VorTIC3 group
Universidad Rey Juan Carlos
Madrid, Spain
carlos.cuesta@urjc.es

Uwe Zdun
Research Group Software Architecture
University of Vienna
Vienna, Austria
uwe.zdun@univie.ac.at

***Abstract*—The Digital Twin (DT) concept has overcome its initial definition based on a purely descriptive approach focusing on modelling physical objects, often using CAD. Today DT often describes a behavioural approach that can simulate an object's dynamics, monitor its state, and control or predict its behaviour. Although DTs are attracting significant attention and offer many advantages in the design of especially cyber-physical systems, most proposals have focused on developing DTs for a specific use case or need without providing a more holistic approach to its design. We aim to propose a domain-agnostic approach for architecting DTs. Here, DTs are directly supported by Domain-Driven Design's notion of Bounded Contexts (BCs), hiding all the domain-inherent specifications behind BC boundaries. These BCs are also the central abstraction in many microservice architectures and can be used to describe DTs. A Wind Turbine DT architecture is used as a running example to describe how every relevant DT property can be satisfied following our proposal for architecting digital twins. A qualitative evaluation of this case by five external practitioners shows that our DDD-based proposal consistently outperforms the 5-dimension model used as the reference approach.**

***Index Terms*—Digital Twin, Domain-Driven Design, Hexagonal Architecture, Microservice, Bounded Context, Design Science**

I. Introduction

Two decades have elapsed since Grieves and Vicker [1] coined the term *Digital Twin* (DT) in 2002 as the conjunction of three elements: "a real space, a virtual space, and a link for the bi-directional flow of data between both spaces for the convergence of virtual and physical objects". The DT concept has developed from its initial definition, which provided a purely *descriptive approach* that typically focuses on modelling physical objects, often using CAD. Today, it often refers to a *behavioural approach* able to simulate their dynamics, monitor their state, and control or predict their behaviour. Therefore, DTs are different from what has been called in the literature *Digital Model* (DM) and *Digital Shadow* (DS) [2] (see Fig. 1). A DM provides a digital representation of an (existing or planned) Physical Object (PO) that usually offers simulation facilities without automatic data flow. On the other hand, a DS offers an automatic data flow from a PO to a Logical Object (LO), focusing on monitoring the PO and predicting its future state. However, a DT supports a bidirectional data flow between the PO and the LO to provide facilities to control the PO and augment it with new capabilities. Kritzinger et al. [2] argue that most of the proposals up to date have focused their efforts on DM and DS, while there is still a clear need for actual DT proposals.

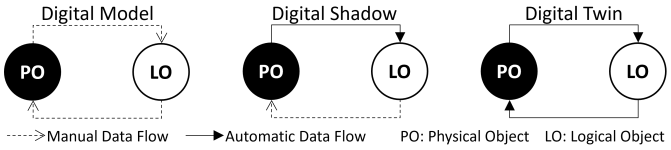

Fig. 1. Digital Model, Digital Shadow and Digital Twin (adapted from [2])

Emerging technologies such as IoT, Edge Computing, Virtualization and Cloud Computing have undoubtedly promoted the community's interest in exploiting DTs as a promising approach to developing future complex systems [3]. For instance, when searching the term "Digital Twin" in the Scopus Search Engine, the references rise from 7 in 2003 to 8,863 in 2022. DTs have been used in many different domains such as Manufacturing [2], Smart Farming [4], Smart Cities [5], Healthcare [6], etc.

Although DTs are a promising approach for developing complex systems [7], most of the proposals developed during these years focus on developing DTs for a specific use case or need, without providing a more general approach that guides their design and development. This need has already been stated in the literature. For instance, Kamble et al. [8] claim that a holistic approach, not just restricted to the local manufacturing systems domain, is required. Minerva et al. [9] state that there is a lack of modelling approaches that help to make the DT concept a reality. In a literature review [10], it is concluded that a unified DT modelling framework is urgently needed, being one of the most promising research directions in the field of DT. This is the motivation of our work: to offer a domain-agnostic approach to the design of DT.

To a certain extent, the notion of DT is yet another expression of the classic design rule of *direct mapping*[11], i.e., the idea that there should be a direct correspondence between concepts in the problem domain and entities in the solution space. This is also a main driver behind *Domain-Driven Design* (DDD) [12]. This approach has

* This paper is part of the R+D+i project PID2019-108915RBI00 funded by MCIN/AEI/10.13039/501100011033.
† Attendance to ICSA2023 funded by Google in the context of the ACM-W Scholarship. †† Corresponding author.

become popular during the last two decades and continued evolving ever since [13]. It defines a dual (strategic and tactic) process, in which a set of patterns are applied during software design, leading to the construction of a domain-based design structure consistent even within a varying environment. Though the basic conceptual core has remained constant, several important DDD concepts appeared later. This includes the relationship to the *Hexagonal Architecture* a.k.a. *Ports and Adapters* [13], which has a separate origin but has become an integral part of the proposal, replacing the original layered architecture [14]. Not only it provides a better form of dependency inversion, but it has proven to be more suitable for the definition of service-oriented architectures (microservices in particular) and, as we will argue, also of DTs.

All these aspects have led us to define our first Research Question, **RQ1**: *Is it possible to apply* DDD *for architecting DTs to satisfy DT properties?* Then, provided that the answer is positive, **RQ2**: *Does this proposal offers results with higher satisfaction of DT properties than other existing proposals?* There are many conceptual correspondences between both areas, which will be explored in the following. Our description of a Digital Twin is directly supported by the notion of *Bounded Contexts* in DDD: the context in which a certain object is used defines its meaning as a DT. Every (Bounded) Context (BC) might work as an independent system fragment, and their relationships to each other define the final structure: their architecture. These independent BCs are also a core abstraction of *microservices* modelled with DDD: often, BCs or sub-structures of the BCs, such as aggregates, are used to define microservices [15]. We argue that we can also use them to describe DTs.

This work is structured as follows. After this Introduction, Section II presents the Research Methodology applied to conduct this study. Then, Section IV presents the main DT concepts, focusing on the properties they should satisfy. Related work is briefly presented in Section III. Our proposal is described in Section V: how DDD may be applied for architecting DT. Finally, the main results of the evaluation conducted as well as the drawn conclusions and our future work are described in Section VI and Section VIII, respectively.

II. Research Method

The main goal (**RQ1**) of our research study is to investigate whether DDD can be used to architect DTs. Then, our second goal (**RQ2**) is to determine whether the results of our proposal offer a higher satisfaction of DT requirements when compared to other proposals. As RQs focus on the design of novel artefacts, a Design Science research study [16] was conducted, comprising the following five steps:

1) Awareness of the problem: A literature review was conducted to detect whether any proposal offers a holistic approach to architect DT. However, as presented in Section III, the alternatives identified focused on specific domains in their descriptions.
2) Suggestion: We designed the approach presented in Section V, which applies DDD for architecting DTs.
3) Development: We further elaborated on the tentative design by developing a UML profile that facilitates the description of software architecture for DTs. This profile is not presented in this paper but has been used to specify the running example presented in Section V.
4) Evaluation: The running example developed was used to evaluate this proposal, using the properties identified by Minerva et al. [9] as the criteria that a DT should satisfy (see Section IV). A qualitative evaluation was conducted, including a survey to compare the current proposal with the 5-dimension model presented in [17], as explained in Section VI
5) Conclusion: The evaluation results are reviewed regarding the RQ2 being the conclusions presented in Section VI.

III. Related Work

As was stated in Section I, Digital Twins are attracting extraordinary attention from both practitioners and researchers. Most of the proposals developed up to date have focused their efforts on developing DT for solving specific problems. For instance, they have been widely embraced by the industry as a key technology for digital transformation [2], e.g. to predict the performance of the production chain, assess reconfiguration changes, manufacture customized products, etc. They are also being applied in the development of smart cities [5], to monitor the performance of the infrastructures or the human dynamics continuously, simulate different scenarios and control their evolution over time. Another example is their use of personalized medicine [6], to monitor a person's bio-physical and lifestyle information, provide data-driven therapies, preventive care, etc.

Most of these proposals have been carried out from scratch, without following a holistic approach that may be applied in different domains or different systems. One of the main reasons for this is the lack of architectural guidelines. In this context, some of the already developed proposals promote adapting the Reference Architecture Model 4.0 (RAMI 4.0) [18] for modelling DTs. RAMI was initially defined for the Industry 4.0 initiative, to control the lifecycle of any asset that offers value to a business, no matter whether it is a physical product, a software component or a process. The example presented in [19] proposes to adopt the asset administration shell (AAS) concept for modelling the information to be managed by a DT, facilitating interoperability among the different subsystems. However, these proposals focus their efforts on modelling the information that the DT should monitor, instead of providing an integral process for architecting DTs. Recently, ISO published standard 23247:2021 [20] for creating any element that may be observed in a manufacturing process. Although the standard describes a Reference Architecture (RA) for constructing systems using DTs, it mainly focuses on providing facilities for interoperability in the manufacturing domain.

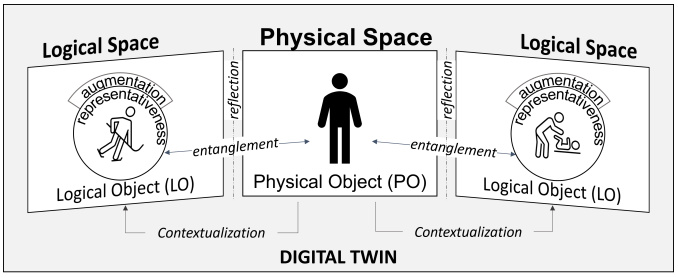

Fig. 2. Digital Twin: Physical Object and Logical Object

Some other works offer guidelines for designing systems, including DTs as one of their constituents. For instance, Kirchhof [21] has presented a model-driven method to design Cyber-Physical Systems and their integration with DT. This is a very promising proposal that focuses on the generation of this integration. However, it has not been defined as a process that guides the design of DTs as reusable units.

Other works have focused on defining alternatives for modelling DTs but on specific domains. One of these is the 5-dimension model [17] that considers, while designing DT, not only the PO, the LO and its connection, but also DT data supporting an accurate information capture and the DT functions to be provided. Similarly, the DTH modelling process [22] has been defined to guide the design of DT in the health domain, outlining the information to be monitored for tracking a patient's physical conditions. These models describe its behaviour and the deduction models for evaluating, reasoning and predicting its needs.

Regarding the use of DDD for designing DT, only a few works have been applied in specific domains. Li et al. [23] have presented an extension of DDD, used to explore the domain of the social manufacturing process and their subsequent implementation with enabling technologies. However, they are not presenting a proposal for architecting DT. Similarly, Telnov et al. [24] propose to apply DDD to identify the different processes during the creation of a product, and to describe how these processes may interact. Still, they are not guiding the design of DT.

IV. Digital Twins: Main Concepts

As mentioned in Section I, there is no standard definition of DT, but many different definitions have been offered in the literature. Just recently, in the manufacturing domain, the ISO/DIS 23247-1 [20] standard has been published that defines a DT as "a fit for purpose digital representation of an observable manufacturing element with synchronization between the element and its digital representation". A more general definition has been provided in [9], where a DT refers to the physical component (PO), the logical component(s) (LO) and the relation between the physical and logical entities. The DT links two different entities, a real one that is relevant in the physical world and a software one that is executed in a virtualization space (see Fig. 2).

Different reasons explain why DTs are considered a game-changer, mainly explained because of the advantages they offer [9]: i) LOs are models of the POs, facilitating their comprehension; ii) LOs are used to store large volumes of information of the POs that may be used for prospective tasks. iii) LOs can be used to trace POs throughout time and space. iv) DTs can be used to detect likely relationships and commonalities between POs. v) DTs may be exploited to manage the life cycle of POs. vi) DTs may be used as services that may be exploited as a business advantage. According to Minerva et al. [9] a DT must satisfy three essential properties:

- (P1) *Representativeness and Contextualization*. Representativeness means that the LO must mimic the status and features of the PO. Here it is important to highlight that many authors also revolve around the idea of similarity as a cornerstone of the DT, making the development of DT a cumbersome task when such similarity introduces complexity that is finally not needed for the goals of the DT. For this aim, contextualization determines that only the information and features of the PO relevant to the context of use should be considered in the LO. Thus, contextualization is key to developing such LOs as software versions of POs. In this sense, we may consider contextualization as a function of time and space. That is, different LO may be needed depending on the specific location and moment in time. Figure 2 shows that if a DT of a person was created, we would need to represent different information, status, etc., depending on the context where such person is.
- (P2) *Reflection*. This property states that a PO is timely and univocally represented using the values of the attributes, status and behaviour of its LO.
- (P3) *Entanglement*. The LO must receive in real-time (or close to real-time, depending on the context of use) the information that represents the PO so that this information can be made available to other applications and services. It is worth mentioning that this entanglement can be: *strong* when the communication PO-LO is bidirectional and real-time; *simple* when it is either real-time or bidirectional; *weak* when values of the PO are inferred.

There is another group of properties that add value to a DT and should be considered when it is developed [9]:

- (P4) *Augmentation* means that the LO can enhance the PO by augmenting its functions and features. This is achieved by exposing an API that facilitates the management of the DT.
- (P5) *Composability* refers to the ability to group LOs into a composed one so that both the individual and composed objects can be observed and controlled. Just consider the Wind Turbine example illustrated in Fig. 4. It comprises different elements such as a Rotor, a Gearbox, etc. The Wind Turbine could be considered a single PO, but it may also be considered a composition of single POs. The latter is more powerful because it represents each one of such single POs as LOs that may be individually managed. Moreover, it would also be possible to control the whole system

by abstracting away only relevant properties.

- (P6) *Memorization* refers to the ability to store all DT's meaningful past and present data and the context of when and where such data originated for their later analysis.
- (P7) *Predictability* means that an LO of a DT may be used to simulate its behaviour and interaction with other DTs to determine the outcomes in a likely future or context.
- (P8) *Replication* refers to a PO being replicated several times using different LOs in a virtual space. Moreover, a LO can also be replicated according to the needs of a specific application. This is the general ability to replicate an object in a different environment. A PO can be virtualized and replicated several times in a virtual space.
- (P9) *Persistency* says that the LO should be created to support that the DT is available over longer periods.
- (P10) *Accountability/Manageability.* A LO should provide facilities to deal with damages or problems of both its PO and itself.
- (P11) *Ownership* states the need to establish the ownership and usage rights both of the DT, especially of the LO, and the data it produces.
- (P12) *Servitization* refers to the ability to offer DT functionalities and processes as a new service in the market.

All these properties, except for (P11) and (P12), have been considered for defining our proposal, as detailed in Section V. Properties (P11) Ownership and (P12) Servitization have not been considered in this work as they do not apply to the capabilities or functions of a DT.

V. A DDD-based proposal for architecting DT

This section presents our proposal to guide the process of architecting DTs. As already noted, the main inspiration is to apply concepts from DDD [12] [13], adapting them to DT design and using them to support stakeholders during its application.

Our proposal for a DT-architecting process is primarily driven by DT properties and their comparison to DDD concepts; essentially, the definition of a *bounded context* becomes the entry point to this process.

As illustrated in Fig.3, the process entails two main activities: *Strategic Design*, which is presented in Section V-A, and *Tactic Design*, described in Section V-B. The architecting of the DT for a Wind Turbine (WT) was realized using the requirements described in [17]. Some of the diagrams specified are used in the following to illustrate the proposal, so many details required for a real-world Wind Turbine have been omitted. The reason to select this system was the comparison in the evaluation process, as explained in Section VI.

A. Strategic Design of DTs

At the core of DDD is the distinction between *strategic* and *tactic* design. Evans' [12] strategic design mainly affects broader, e.g. infrastructural and architectural, concerns. Therefore, as depicted in Fig. 3, our process begins with a *Strategic Design* activity, which is defined as the iteration of three tasks.

Identify the DTs. To properly identify DTs, which are a combination of PO, LO and the communication among them, two properties are key: (P1) *Contextualization* and (P5) *Composability.* That is, first (a) we should consider which are the likely contexts of use, regarding time and space, of our DT; so that we identify a different DT for each one of such contexts. Next, (b) we should determine whether such DTs should be considered as a single DT, or a composite one, depending on the control level required. In this activity, we simply start by enumerating the different POs in our system. After that we identify which ones are going to be considered as separate DTs and which ones are not. Then we are ready to *abstract* this "LOs map" into a proper model in the next step.

Considering the Wind Turbine (WT) example, only one context was considered for optimizing its operation. Fig. 4 (b) illustrates the different POs of the WT as well as the relationships among them that have been identified. It should be mentioned that this design was conducted in two different cycles. During the first cycle, every sensor was also identified as a relevant PO. However, we noticed that this decision added an unnecessary level of complexity. Therefore, they were abstracted out from the model and considered just as properties of the POs.

Map Types of DTs to Bounded Contexts. The goal of this whole process is to describe the architecture of DTs. Therefore, at the design level, we are not interested in individual DT instances, but in identifying the different *types of DTs* (tDT). That is, the process must distil the different kinds of *distinct* DTs which are present in our system. This is relevant because we are designing the software comprising these different types, even when there could be many instances of them in a specific situation. This "instance model" will compose a *concrete* configuration of our architecture.

One of the main concepts in strategic DDD is the *Bounded Context* (BC). According to Millet [13], a Bounded Context is a "semantic contextual boundary. This means that within the boundary, each component of the software model has a specific meaning and does specific things. The components inside a Bounded Context are "context-specific and semantically motivated," facilitating that the model keeps consistent and meaningful. We consider this concept to be critical while designing DTs due to one of its properties [13]: they should be *loosely coupled* to allow them to be developed in isolation. Defining *types* of DTs as these loosely coupled Bounded Contexts means that they may be developed and evolved independently and reused for different applications, avoiding creating silos of functionality. Therefore, every *type of Digital Twin* gets mapped to a specific *Bounded Context*. In fact, in this process we use these terms interchangeably, as both abstractions are functionally congruent, i.e. ($tDT \cong BC$).

Considering the Wind Turbine example, Fig. 4(c) shows

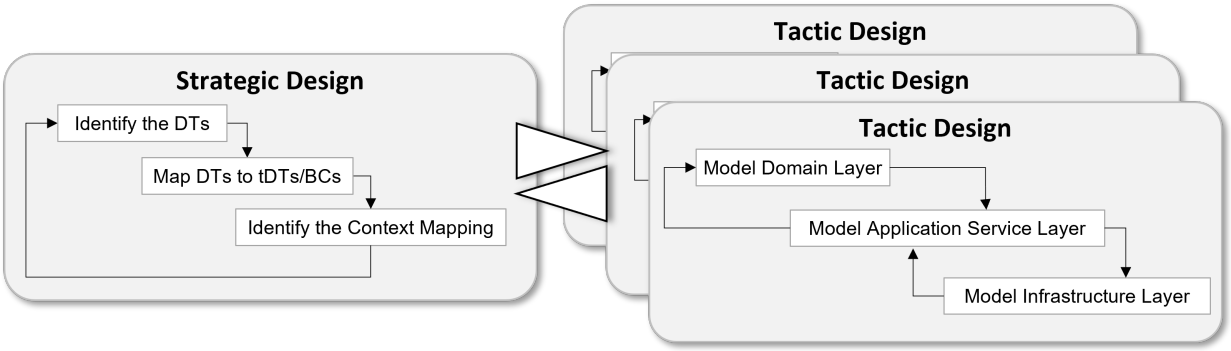

Fig. 3. A DDD-inspired process for architecting DT

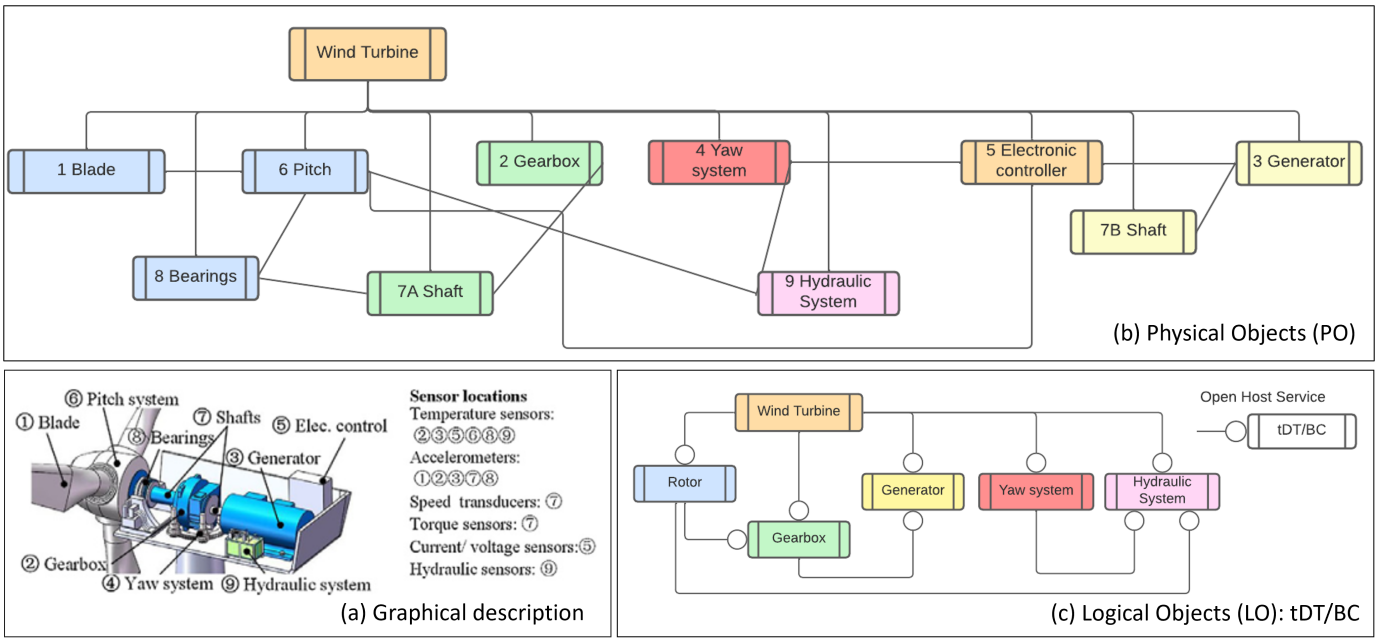

Fig. 4. Strategic Design for a Wind Turbine: (a) graphical description (extracted from [17]); (b) POs identified; (c) types of DTs and Context Mapping (different colours are used to show the mapping between POs and tDTs)

the tDTs that were finally identified. Most of these tDTs subsume more than a PO from those previously identified. For instance, Rotor subsumes the Pitch, Blade and Bearing POs, and WindTurbine subsumes both WindTurbine and Electronic Controller.

Identify the Context Mapping. Considering that different BCs, i.e. different tDTs, could be identified in the previous step, they will need to interact among them to support the required (P5) *Composability*. This interaction must also be modelled, and the DDD concept of *Context Mapping* [13] also serves this same goal. This mapping is used to identify the integration points between Bounded Contexts and the data flow between them. For this purpose, contracts and boundaries between BCs must be clearly defined to facilitate their interaction and evolution over time. According to DDD guidelines, the relationship between them and their type must be specified for every two Bounded Contexts that need to interact. They propose organizational and *technical* types for such integration [13]. In this work, the following three types of DDD-based technical relationships are suggested:

- *Open Host Service*. It is suggested that this should be the default type of relationship between any two BCs, as DTs should satisfy the (P4) *Augmentation* property. This type of relationship determines that Bounded Contexts expose their functionality as a set of services through a public API, precisely that property's definition. Fig. 4 (c) shows that this was the relationship established among the different tDT/BC identified in the Wind Turbine example.
- *Shared Kernel*. Whenever two tDTs (i.e., BCs) share many concepts and functionality, these commonalities can be extracted as a shared model that is known as *Shared Kernel*. This type of DDD relationship would facilitate satisfying both the (P1) *representativeness* and (P5) *composability* properties, which have already been considered.
  For example, consider that a Human DT (HDT, see Fig. 2) is being designed. Different DTs may have been identified because different contexts (e.g. health,

play) were detected. However, while designing such HDT, we may also detect that different sensors (e.g. accelerometer, heart rate) may be applied in specific contexts. This could lead to the identification of new DTs, as, in the end, those sensors are actually POs, and their corresponding LOs may be created to mimic their status and features, as requested by (P1). But, these new sensor-based DTs are actually sharing many concepts and information with prior DTs in the compound HDT, so they would have a *shared kernel* relationship, satisfying the (P5) composability property.

- *Anticorruption Layer (ACL).* Whenever a DT needs to communicate with legacy or third-party systems, then an *Anticorruption Layer* relationship should be created to protect the integrity and autonomy of this DT. An ACL is used for integration purposes so that it is responsible for translating between both the BC it belongs to and the BC to which it is related. Thus, it will be responsible for the bidirectional translation between the DT and third-party systems.

B. Tactic Design of DT

Once the main contexts have been identified and their boundaries outlined, strategy has defined the global scope; the actual enactment into concrete elements is the domain of *tactics*. In the context of DDD, *tactic design* defines the set of *domain* elements (entities, services, etc.) which compose a Bounded Context, and it takes into account specific low-level operational concerns, such as concurrency. For instance, aggregates are conceived to act as transactional consistency boundaries [13].

Still, tactics must work within the limits defined by strategy. But this is not a modular decomposition in which strategic design defines the BCs, and then tactic design develops their contents. Actually, both design approaches work together to build the desired system: as strategy defines boundaries and relationships, tactics describe how subsystems enforce these limits while enabling these interactions. Unsurprisingly, the connection between both perspectives is *architecture*; the original (tactical) pattern for this purpose was Layered Architecture [12]; but as already noted, Hexagonal Architecture has acquired this role [13].

The main outcome of Strategic Design is the set of DTs to be developed and the different Context Mappings (CMs) among them. Then, Tactic Design intends to identify the *Application Architecture* of these DTs. Actually DDD does not enforce a specific style to architect the application, but it *does* require it to be chosen to protect the integrity of the *domain models* enclosed within each Bounded Context. For this purpose, the separation of concerns is considered critical: the architecture separates the technical details from the complexities of the domain models. This architecture must also serve to encapsulate the low-level details of the domain model of every Bounded Context, to protect clients from any potential changes in their logic.

Fig. 5 illustrates the internal architecture of the BC defining a DT. At the centre of this architecture is the *Domain Layer* that captures the logic of the domain. Then, the *Application Layer* abstracts such logic domain behind an API, to publish the relevant business use case. Both layers are isolated and protected against changes, unnecessary dependencies from clients, or infrastructural issues using the *Infrastructure Layer*. To satisfy these requirements, we could use a variety of alternatives. However, Fig. 5 already presents them using the approach of the aforementioned Hexagonal (a.k.a. Ports-and-Adapters, a.k.a. Onion) Architecture, which has become the recommended style [13]. This approach eases the required separation of concerns, structuring the application into different areas that change together. It is worth noting that, to enforce a separation of concerns, one of the cornerstones of this style is the use of *Dependency Inversion* [12]. This concept promotes that those components offering low-level services, as Infrastructure does, implement the interfaces defined by high-level components, such as the Application Layer. Thus, the domain layer does not depend on anything else, and the application service layer just depends on the domain layer. That is, dependencies between layers are always *directed* towards the centre of the architecture. Thereby, this work proposes that for every DT (as a Bounded Context) identified during Strategic Design, its architecture is specified in the Hexagonal Architecture style.

As noted in Fig. 5, each one of the properties that a DT may satisfy (refer to Section IV) has been traced to a specific layer. Thus Fig. 5 may be used as a *decision map* to decide which ones are going to be considered in a specific implementation: depending on which properties are expected from the DT being modelled, specific decisions must be made in the affected layers. The remainder of this section presents how each of these layers should be modelled and how the DT properties are addressed.

Model the *Domain Layer*. In this layer, the domain model is specified to satisfy the use case of the DT. Thus, this domain model is key to satisfying the (P1) *representativeness* and (P2) *reflection* properties of DT. It must be highlighted that this modelling must follow a technology-agnostic approach, focusing just on the DT's rules, logic and workflows, without introducing any technological detail. While modelling such domain concepts, the following DDD concepts may be used:

- *Entity.* Each concept can be univocally identified among all other concepts of the DT's domain model. It may be mutable over time according to our needs, that is, its state may change. As Fig. 6 shows, Electronic Controller was initially identified as a PO of the Wind Turbine. It is now specified as an Entity used to define the control policy responsible for WT productivity.
- *Value Object.* It describes every concept that is immutable in the DT. They are used to specify the properties of an Entity. As an example, in Fig. 6 Power was defined as Value Object to represent the

DT Properties	DDD concepts
(P1) Representativeness & Contextualization	Entity Value Object Aggregate
(P2) Reflection	
(P3) Entanglement	Event Sourcing
(P4) Augmentation	Open Host Service
(P5) Composability	Context Mapping Messaging
(P6) Memorization	Event Sourcing Repository
(P7) Predictability	Domain Services Event Sourcing Integration Services
(P8) Replication	Integration Services
(P9) Persistency	Event Sourcing Repository
(P10) Accountability & Manageability	Open Host Service

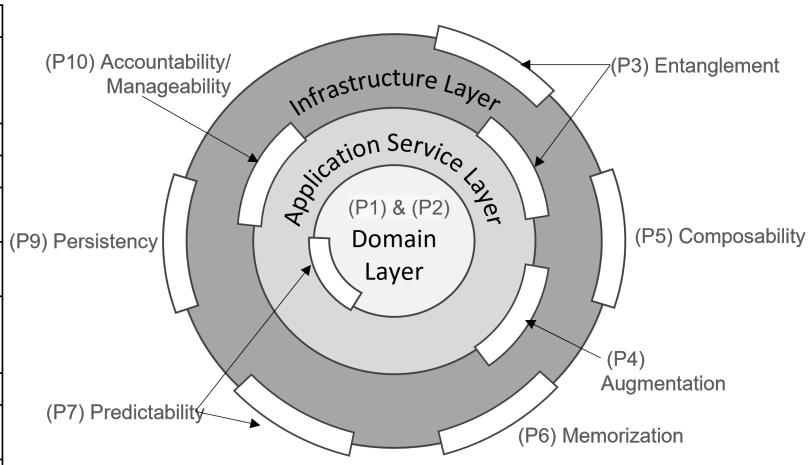

Fig. 5. Hexagonal Architecture: Tracing DT properties

metric Electric Power.

- *Aggregate*. It is defined as a composition of one or more Entities and Value Objects. An Aggregate is used to maintain *transaction consistency* within a DT, that is, to maintain which business rules must be satisfied once a transaction ends. The design of Aggregates is a popular topic in the literature, and several guidelines are provided [12] [13], but these are left out of this work due to space constraints. Fig. 6 shows that WindTurbine was specified as an Aggregate that facilitates referring to WindTurbine and ElectronicController as a single concept.

While modelling the domain layer, *Domain Services* must also be specified. These are used to represent the *behaviour* of the DT using Entities, Value Objects and Aggregates, as well as relying on the Application Service Layer for any technical detail. For instance, if a domain service requests to retrieve any information from a database, the Application Service Layer will be in charge of this task due to dependency inversion. The (P7) *predictability* property can be satisfied using these domain services. It is often requested to simulate the dynamic of DTs to predict their outcome in specific situations and environments. To conduct such a simulation, it is necessary to have intrinsic knowledge of the domain: Entities, Value Objects and Aggregates must be used. When such simulations are defined as domain services, the rules governing the DT's likely behaviour can evolve without affecting their clients. Fig. 6 depicts the SimulatorWT domain services that provide three different types of analysis: deformation, stress and crack. As can be observed, dependency inversion has been used to facilitate simulation services' evolution from Finite State Machines to any other approach, such as Petri Nets, or stochastic models.

Model the *Application Service Layer*. Following the hexagonal style, this layer is responsible for providing the DT's *use cases*, by delegation to the other two layers, due to dependency inversion. Let's consider the WT example again, as presented in Fig. 6. There, *application services* may be offered as business use cases, such as monitoring the power generated by a Wind Turbine, starting or stopping its operation and predicting its fault. To do that, this Layer orchestrates and coordinates the different steps comprising each one of those use cases. So the Application Service Layer is in charge of retrieving the domain objects stored, e.g. in the database, using *infrastructure services* (in the Infrastructure Layer), and delegating to these objects, and to *domain services* (in the Domain Layer), the responsibility of making the final decision regarding the specific business use case at hand. Therefore, every *service* that a DT offers to its clients is published in this layer, hiding the details of the Domain Layer. Thus, it is responsible for satisfying the (P4) *augmentation* property, which exposes the DT functionality as a public API.

Another important feature of this layer is to notify about *changes* in the status of domain objects that may affect other system elements. As explained in Section IV, the (P3) *entanglement* property is related to the communication between the PO and the LO, which must be *bi-directional* to define a proper, full-fledged DT. Therefore, this layer is responsible for notifying the POs about any change in their LOs. For this purpose, the Application Service Layer relies on *Event Sourcing* services, offered by the Infrastructure Layer (as seen next).

Finally, another important aspect that may be contemplated during the DT's design is (P10) *accountability & manageability*, providing the means to deal with potential problems or damages. When this feature is considered, this Layer would be responsible for providing facilities to deal with such damages, as part of the public API. For instance, as Fig. 6 shows, different services have been defined for predicting faults and maintaining a WT.

Model the *Infrastructure Layer*. This layer is responsible for all the *technical* capabilities that the DT architecture might have. Thus, Domain and Application layers need just to focus on modelling the DT's behaviour and use cases, respectively. In contrast, the Infrastructure Layer strives with technical details that do not impact these use cases but only on their support. As summarized

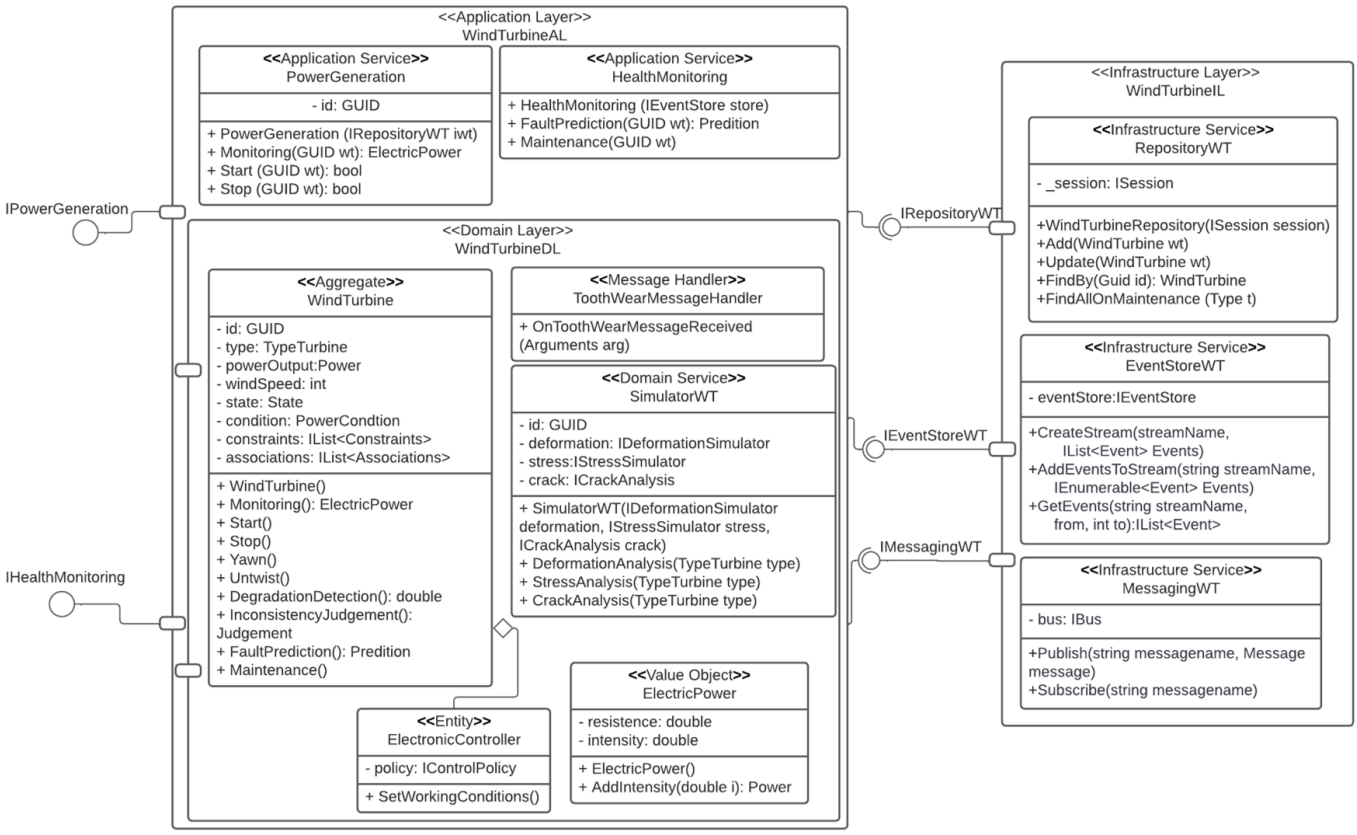

Fig. 6. Tactic design for the Wind Turbine DT

in Fig. 5, different DT properties have been traced to different DDD concepts, which are supported in this layer in the form of services. Each one of these *infrastructure services*, along with the DT properties they support, are described in the following:

- *Messaging*. As stated above, whenever a DT is being designed, it should be considered a reusable element that may be exploited for different purposes. Therefore, a DT must be developed as a *loosely coupled* unit to evolve independently from other elements. Also, DDD guidelines [13] suggest that a *messaging system* should be included in the Infrastructure Layer. This would provide support for (P5) *composability* because, as DTs are completely decoupled from each other, *composite DTs* may be created as necessary, without affecting the *component DTs*. Besides, in case any component DT fails, the remaining DT would not be affected. For instance, the gearbox is one of the most failure-prone elements of a Wind Turbine. This motivated us to specify this element as a tDT to facilitate its management (see Fig. 4). Therefore, a message handler has been defined to allow the WindTurbine to react promptly to any message informing about a fault of the gearbox (e.g. tooth wear).
- *Repository*. An important DT property is (P6) *memorization*. This refers to the ability to store relevant information about the DT. In DDD, one of the most popular services that the Infrastructure layer could offer to persist any domain object is the Repository. This acts as a façade that removes the complexity of some persistence frameworks and maintains a clear-cut difference between the *domain model* and the *data model* used to interact with clients. Therefore, introducing it during a DT's design would make it possible to store all relevant data in the DT, providing it with the (P6) memorization property. Moreover, this service also serves to satisfy the (P9) *persistency* property. As the repository can maintain all meaningful data and the PO status, it would also make the LO persistent and resilient over time. Fig. 6 illustrates that RepositoryWT has been defined to maintain meaningful data of a Wind Turbine, such as its type, constraints regarding wind conditions for its operation, etc.
- *Event Sourcing*. The introduction of this facility in the design provides a noteworthy advantage, as it is responsible for chronologically storing the sequence of events emitted by a DT. Therefore it explains why and how this has reached a specific status. This provides support to satisfy at least three different DT properties. First, (P3) *entanglement* between the PO and LO is achieved by using whichever event service is supported by the PO. Second, (P6) *memorization*, as it helps capture the DT's behaviour. Third, (P7) *predictability*, as analysing past sequences of events may help to predict how a DT could behave in the future.

Finally, (P9) *persistency*, because in the case of failure of a PO, the sequence of events may be analysed, to determine how to reset it to an acceptable state. As shown in Fig. 6, EventStoreWT has been defined to collect all the events that may affect the Wind Turbine, for instance, the wind speed.

- *Integration Services.* The Infrastructure layer may also offer facilities to integrate other systems or applications, which have no impact on the design of the domain model or business use cases but can support different technical details. For instance, these integration services may be used to support (P7) *predictability*. In fact, during a DT's design, it is rather frequent to use external testbeds or experimentation settings, to simulate the behaviour and interaction of this DT with other elements. Including this type of service in the Infrastructure, the layer would help to maintain both the Application Service and Domain layers agnostic from these technologies so that they could be modified or replaced at convenience.

Finally, among Minerva's DT properties, only (P8) *replication* has yet to be explicitly addressed. This is a complex issue, as it refers to individual DT *replicas*, and the relationships between them; so this is located at the *configuration* level, rather than at the *architecture* level. A connection between two *same*-DT replicas cannot be expressed in their *type* definition. There is some room to consider that property. When a certain pattern (e.g. "master replica") is likely to appear, certainly the tDT might be *prepared* to support such a construction. Even more, existing frameworks (e.g. containerization) often provide facilities to manage these arrangements out of the box. This sort of delegation can therefore be achieved using *integration services*, again.

In summary, tactic design can be used as a sort of decision tree, in which every desired DT property can be mapped to a certain architectural construction; the resulting structure would depend on the set of chosen capabilities.

VI. Evaluation

As outlined in Section II, our proposal was evaluated to answer **RQ2** by doing the following steps:

1) *Selecting a reference proposal.* An expert practitioner in DT, not involved in the definition of this proposal, was provided with the list of DT properties [9] and was requested to select the best proposal for architecting DT available in the literature that presents a detailed description of a real system. As a result of this step, the 5-dimension model proposal presented in [17] was selected.
2) *Architecting the Wind Turbine example.* We authors applied the presented proposal for architecting the WT. The main outcomes have already been presented in Fig. 4 and Fig. 6.
3) *Collecting the evaluations.* Different practitioners were contacted by email with the following instructions. First, they should read carefully the 5-dimension model proposal, as presented in [17], and then our DDD-based proposal, paying special attention to the process of architecting a DT. Second, they were requested to fill in an anonymous survey structured in the following sections: i) demographic data; ii) a section for each analysed proposal that had 10 questions in the form: "*As a practitioner, I found that applying the proposal [PXXX], the outcome satisfies the property [DTPY]*", where [PXXX] is the analysed proposal and [DTPY] each one of the DT properties; iii) any general comment they would like to provide us with.

Fig. 7 summarizes the answers given by the five solicited subjects. All of them have more than 5 years of experience in Software Engineering. This was a compulsory requirement to ensure they have the background to make a fair comparison between the proposals. Subjects 1 and 2, as experts, have developed systems using DT. Subjects 3 and 4 have previous backgrounds in the field. Finally, Subject 5 has not worked with DTs but is an expert on other related topics, such as IoT and CPS. Regarding DDD, only Subject 3 is an expert; the other subjects just have some background regarding DDD or related concepts, such as hexagonal architecture. This aspect is highly relevant, as a bias might have been introduced if most subjects were experts in DDD.

Regarding the evaluation outcome, Fig. 7 illustrates that our DDD-based proposal outperforms the 5-dimension model proposal [17] for all the analysed properties. Regarding the mandatory properties (P1, P2, P3), all the subjects considered (*strongly agree*, most of them) that our DDD-based proposal facilitates their satisfaction. Only (P2) Reflection was evaluated as *neutral* by Subject 1. Even when the other subjects evaluated this property positively, this suggests that our future work should be oriented to guide architects in defining the domain layer. Regarding the 5-dimension model proposal, subjects also evaluated these 3 properties positively. These results are consistent, as the 5-dimension model proposal is highly

	Subject 1		Subject 2		Subject 3		Subject 4		Subject 5		Average	
Experience in SE (years)	5-10		5-10		>20		5-10		5-10			
Experience in DT	Expert		Expert		Advanced		Advanced		Beginner			
Experience in DDD	Novice		Beginner		Expert		Advanced		Advanced			
DT Properties	**5 dim**	**DDD**	**5 dim**	**DDD**	**5 dim**	**DDD**	**5 dim**	**DDD**	**5 dim**	**DDD**	**5 dim**	**DDD**
(P1) Representativeness & Contextualization	1	2	1	2	1	2	2	2	1	1	1,2	1,8
(P2) Reflection	1	0	1	2	1	1	2	2	1	2	1,2	1,4
(P3) Entanglement	2	1	0	2	1	2	2	2	0	1	1	1,6
(P4) Augmentation	0	1	1	2	-1	2	0	2	0	1	0	1,6
(P5) Composability	-1	2	-2	2	0	2	1	2	1	2	-0,2	2
(P6) Memorization	1	1	1	2	1	2	2	2	0	2	1	1,8
(P7) Predictability	1	-1	1	2	1	1	2	2	1	2	1,2	1,2
(P8) Replication	0	0	-2	0	0	1	1	1	1	2	0	0,8
(P9) Persistency	1	1	1	2	1	1	2	2	1	2	1,2	1,6
(P10) Accountability & Manageability	1	2	-2	2	1	1	2	2	1	1	0,6	1,6

2: Strongly Agree **1**: Agree **0**: Neutral **-1**: Disagree **-2:** Strongly Disagree

Fig. 7. Evaluation of the 5-dimension model (5 dim [17]) and this proposal (DDD))

detailed regarding describing the physical objects and their mapping to logical objects.

However, greater differences can be perceived while analysing the other properties. As can be observed, our DDD-based proposal facilitates the satisfaction of all these properties. The worst score (0,8) was obtained regarding (P8) Replication. We consider that this was probably due to the absence of constraints for this aspect in the presented example. Regarding (P7) Predictability, we can notice that both proposals had the same score. This is highly positive because although the 5-dimension model focuses its efforts on this aspect, it does not outperform our DDD-based proposal. We have also detected that Subject 1 evaluated this property negatively. Thus, some additional work should be considered on this aspect. We plan to investigate whether some alternative approaches for modeling predictability could be considered; for instance, using aggregations or entities instead of domain services. For (P9) Persistency and (P4) Augmentation, we also notice consistent results regarding our DDD-based proposal. Subjects perceive that the LO developed with our proposal not only facilitates additional functionalities for the PO but also improves its availability over time. For (P5) Composability, (P6) Memorization and (P10) Accountability & Manageability, our DDD-based proposal clearly provides much higher scores, where all the subjects *agree* or *strongly agree* regarding the satisfaction of these properties.

VII. Threats to Validity

In this section, we briefly discuss the threats to validity of the presented evaluation.

Construct validity. These threats can be considered mitigated. First, the paper itself presents the conceptual framework (i.e., Minerva's Properties from Section IV) to be considered, so the context is clearly delimited. Also, the survey is unequivocally asking for RQ2 itself.

Internal validity. The greatest threat may be due to the selection of the subjects (i.e., participants in the evaluation), and the bias which could be derived from this. Different decisions were made to mitigate this threat. First, all of them are practitioners, with an industrial rather than an academic background; the enforced requirements were seasoned expertise in software engineering and familiarity with modelling concepts. Most of them are familiar with DT concepts, granting even more independence when assessing these models. Regarding DDD, we have one expert and four novices in the topic. Thus our proposal should be fairly evaluated in its own merits. Finally, all of the practitioners were fully independent, and there is no contact or connection between them, so a selection-maturation threat is unlikely.

External validity. Although the number of subjects is rather limited, they were chosen as representatives from their own backgrounds. To improve the external validity, the evaluation was conducted by comparing our proposal with the well-known Wind Turbine case study from [17], which can already be considered a reference example in the field itself. Finally, instead of using a self-developed conceptual model for our comparison, we rely on the reference framework by Minerva et al. [9].

Reliability. Again, the main threat here is whether the selection of the subjects might be affected by their relationship to the authors; indeed, practitioners were contacted by the conductors of the survey, directly or indirectly. However, none of them has a working relationship with any of the authors, and they are completely unrelated to most of them. Moreover, no incentive was provided to answer the survey, and all further interaction was fully anonymous. Another threat is related to the selection of the 5-dimension model as reference proposal conducted by an expert practitioner. To mitigate this threat two actions were conducted. First, this expert was not involved in the definition of this proposal. Second, this expert was requested to select the best possible proposal considering it must be highly cited in the literature and present a clear case study, being both conditions clearly satisfied by the 5-dimension model.

VIII. Conclusions and Future Work

Several proposals are already intended to help define and model DTs. They consists of mapping a specific physical configuration, and then its virtual correspondence (i.e. logical objects and their connections), rather than assisting in developing the DT itself. For this reason, many of them rely on describing a pre-fixed software architecture, instead of helping in the design of a suitable architecture for this DT.

The goal of our proposal is explicitly to assist in the design of DTs *able to be reused in different contexts*. The DT architecture must include the relevant abstractions, independently from particular configurations. Therefore, it must identify the corresponding *types of DTs* (tDT), their internal structure and their relationships to each other. This process provides the identification method, the description of their internal architecture and domain model, and the connection pattern. The proposal was used to specify a Wind Turbine. After that, we are in the position to answer positively **(RQ1)**, that is, DDD can actually be applied to the design of DTs. Finally, the proposal was evaluated by external practitioners who, after reading both the 5-dimension model proposal presented in [17] and our proposal, answered a survey to evaluate to which extent each proposal supports Minerva's properties. The outcome satisfies all mandatory DT properties, and also supports most of the optional ones. Thus, we also answer positively **(RQ2)** that is, it provides a higher satisfaction of those properties than the alternative proposal. A 1-to-1 comparison to this other proposal still favours our proposal, even using the case proposed in [17].

Soon, we intend to apply this process to design a full-fledged DT architecture in the healthcare domain, to check its actual impact on real-world development and to obtain feedback from more practitioners. We also plan to conduct a formal case-study evaluation to identify the main issues in the existing process, refine it, and analyse its usability.

References

[1] M. Grieves and J. Vickers, "Digital twin: Mitigating unpredictable, undesirable emergent behavior in complex systems," in *Transdisciplinary perspectives on complex systems*. Springer, 2017, pp. 85–113.

[2] W. Kritzinger, M. Karner, G. Traar, J. Henjes, and W. Sihn, "Digital twin in manufacturing: A categorical literature review and classification," *IFAC-PapersOnLine*, vol. 51, no. 11, pp. 1016–1022, 2018.

[3] A. Niati, C. Selma, D. Tamzalit, H. Bruneliere, N. Mebarki, and O. Cardin, "Towards a digital twin for cyber-physical production systems: a multi-paradigm modeling approach in the postal industry," in *23rd ACM/IEEE Int. Conf. on Model Driven Engineering Languages and Systems: Companion Proceedings*, 2020, pp. 1–7.

[4] C. Verdouw, B. Tekinerdogan, A. Beulens, and S. Wolfert, "Digital twins in smart farming," *Agricultural Systems*, vol. 189, p. 103046, 2021.

[5] N. Mohammadi and J. E. Taylor, "Smart city digital twins," in *2017 IEEE Symposium Series on Computational Intelligence (SSCI)*, vol. 27, 2017.

[6] K. Bruynseels, F. Santoni de Sio, and J. Van den Hoven, "Digital twins in health care: ethical implications of an emerging engineering paradigm," *Frontiers in genetics*, p. 31, 2018.

[7] A. Macías and E. Navarro, "Paradigms for the conceptualization of cyber-physical-social-thinking hyperspace: A thematic synthesis," *Journal of Ambient Intelligence and Smart Environments*, vol. 14, no. 4, pp. 285–316, 2022.

[8] S. S. Kamble, A. Gunasekaran, H. Parekh, V. Mani, A. Belhadi, and R. Sharma, "Digital twin for sustainable manufacturing supply chains: Current trends, future perspectives, and an implementation framework," *Technological Forecasting and Social Change*, vol. 176, p. 121448, 2022.

[9] R. Minerva, G. M. Lee, and N. Crespi, "Digital Twin in the IoT Context: A Survey on Technical Features, Scenarios, and Architectural Models," *Proceedings of the IEEE*, vol. 108, no. 10, pp. 1785–1824, oct 2020.

[10] F. Tao, H. Zhang, A. Liu, and A. Y. Nee, "Digital twin in industry: State-of-the-art," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 4, pp. 2405–2415, 2018.

[11] B. Meyer, *Object-oriented software construction*. Prentice hall Englewood Cliffs, 1997, vol. 2.

[12] E. J. Evans, *Domain-driven design: tackling complexity in the heart of software*. Addison-Wesley Professional, 2004.

[13] S. Millett and N. Tune, *Patterns, principles, and practices of domain-driven design*. John Wiley & Sons, 2015.

[14] L. Bass, P. Clements, and R. Kazman, *Software architecture in practice*. Addison-Wesley Professional, 2003.

[15] A. Singjai, U. Zdun, and O. Zimmermann, "Practitioner views on the interrelation of microservice apis and domain-driven design: A grey literature study based on grounded theory," in *18th IEEE International Conference On Software Architecture (ICSA 2021)*, 2021.

[16] R. J. Wieringa, *Design science methodology for information systems and software engineering*. Springer, 2014.

[17] F. Tao, M. Zhang, Y. Liu, and A. Y. Nee, "Digital twin driven prognostics and health management for complex equipment," *Cirp Annals*, vol. 67, no. 1, pp. 169–172, 2018.

[18] Phoenix Contact, "RAMI 4.0 and IIRA reference architecture models a question of perspective and focus," 2020, last accessed 1 June 2022.

[19] S. Kosse, O. Vogt, M. Wolf, M. König, and D. Gerhard, "Digital twin framework for enabling serial construction," *Frontiers in Built Environment*, vol. 8, p. 864722, 2022.

[20] I. 23247-1:2021, "Automation systems and integration — digital twin framework for manufacturing — part 1: Overview and general principles," 2021.

[21] J. C. Kirchhof, J. Michael, B. Rumpe, S. Varga, and A. Wortmann, "Model-driven digital twin construction: synthesizing the integration of cyber-physical systems with their information systems," in *23rd ACM/IEEE International Conference on Model Driven Engineering Languages and Systems*, 2020, pp. 90–101.

[22] Y. Liu, L. Zhang, Y. Yang, L. Zhou, L. Ren, F. Wang, R. Liu, Z. Pang, and M. J. Deen, "A novel cloud-based framework for the elderly healthcare services using digital twin," *IEEE access*, vol. 7, pp. 49 088–49 101, 2019.

[23] M. Li, Y. Fu, Q. Chen, and T. Qu, "Blockchain-enabled digital twin collaboration platform for heterogeneous socialized manufacturing resource management," *International Journal of Production Research*, pp. 1–21, 2021.

[24] Y. F. Telnov, V. A. Kazakov, and A. V. Danilov, "Technology for designing innovative processes for creating products and services of a network enterprise using an i4. 0 knowledge-based system," *Business Informatics*, vol. 15, no. 4, pp. 76–92, 2021.