Composable Digital Twins on Digital Twin as a Service Platform

Prasad Talasila[1], Cláudio Gomes[1], Lars B. Vosteen[2], Hannes Iven[2], Martin Leucker[2], Santiago Gil[1], Peter Høgh Mikkelsen[1], Eduard Kamburjan[3] and Peter Gorm Larsen[1]

Abstract

Establishing digital twins is a non-trivial endeavour especially when users face significant challenges in creating them from scratch. Ready availability of re-usable models, data, functions, and tool assets, can help with creation and use of digital twins. A number of frameworks/platforms exist to facilitate creation and use of digital twins. In this paper we propose such a platform to manage digital twin assets, create composable digital twins from re-usable assets and make the digital twins available as a service to other users. The proposed platform supports the management of re-usable assets, storage, provision of compute infrastructure, communication, monitoring and execution tasks. Two case studies are used to demonstrate the capabilities of this platform.

Introduction

Digital Twins (DTs) are used to add value to Cyber-Physical Systems (CPSs) of interest, typically called Physical Twins (PTs). At the heart of a DT is a collection of models describing characteristics of PT, which is used to provide additional services for it. Many such services can be imagined and it is the vision of these capabilities that makes DTs so valuable: the main motivation for DTs is to enable real-time monitoring, analysis, and simulation of a PT. This technology facilitates improved decision-making, predictive maintenance, and optimisation in various industries, including manufacturing[1], healthcare, urban planning, and energy storage[2]. DTs enhance efficiency, reliability, and sustainability by providing a comprehensive understanding of complex systems and supporting data-driven insights. They represent a natural stepping stone from the massive availability of sensors and data in different industries, and it is our conviction that DT architecture is common across many such industries.

However, the implementation of DTs is still a major endeavour, requiring a significant effort from stakeholders with different disciplinary backgrounds. For instance, establishing the communication between the PT and the DT requires potentially knowledge of computer networks and architecture as well as programming skills, while building predictive models of the PT requires potentially knowledge of the corresponding physics domain. Furthermore, in order to promote re-usability, a DT may marshal a collection of data sources and sinks, generic functions, and tools. Delivering such a DT can be a complex task; coordinating and orchestrating the numerous services and models remains a challenge. This is especially true because of the need for several different modelling approaches, including information models, geometry, physics, and behaviours[3,4].

In this work we present the Digital Twin as a Service platform (DTaaS). It aims to both (a) speed up development of DTs and (b) simplify its management during operations based on the notion of *re-usable DT assets*: data, functions, models and other software components that can be reused by multiple DTs. A number of DT platforms have been proposed to reduce the implementation effort in relation to the structural aspects of DTs[5], and some also recognise re-usability as a major challenge, but these platforms that support the creation and execution of DTs from composable components[6,7] enforce certain implementation styles and are not service-oriented. Recently, a similar idea of establishing a DTaaS platform has been proposed[8], but with a primary focus on services related to Augmented Reality.

Our DTaaS platform targets not only re-usability in the design stage of DTs, but also re-usability at operations – different life cycle phases of a DT require different assets. DTaaS gives a practical platform to manage DTs throughout their respective life cycle.

Contribution Our main contributions are (a) a precise definition of DT assets and DT life cycle, (b) an architecture for DT design and operations based on these definitions, and (c) an implementation of this architecture with an evaluation

[1]Centre for Digital Twins, DIGIT, Department of ECE, Aarhus University, Finlandsgade 22, 8200 Aarhus N, Denmark
[2]Institute for Software Engineering and Programming Languages, Universität zu Lübeck, Ratzeburger Allee 160, 23562 Lübeck, Germany
[3]Department of Informatics, University of Oslo, Gaustadalleen 23 B, 0373 Oslo, Norway

Corresponding author:
Prasad Talasila, Centre for Digital Twins, DIGIT, Department of Electrical and Computer Engineering, Aarhus University, Finlandsgade 22, 8200 Aarhus N, Denmark.
Email: prasad.talasila@ece.au.dk

on the incubator[9] DT and firefighter DT of the O5G-N-IoT project[10].

Prior Work This work is an extension of previous work from the co-authors that has introduced the foundations of DTaaS[11], its nomenclature and life cycle phases for DTs deployed in the platform.

This current manuscript introduces the definition of a re-usable DT asset, an extended and adopted architecture to support this definition, and two realistic case studies as an evaluation. The current manuscript also expands on prior work by providing details about how two case studies have gained from establishing DTs using DTaaS. From those case studies, we describe the benefits of re-usable assets, the use of different life cycle phases, the realisation of the DTs and their performance evaluation. Moreover, we provide a more detailed comparison with existing platforms, and introduce two new re-usable commonly used DT services (visualisation and alerts).

Structure The rest of this paper is structured as follows. We first introduce Background and Related Work in DT platforms. Afterwards, we introduce the two different Case Studies (a DT for food fermentation and a DT for rescue mission support in firefighting), which are consequently used as running examples for illustration. Then we give the definitions and illustrations for Re-usable Digital Twin Assets, Digital Twin Definition and Phases in Digital Twin Life Cycle, before we give an overview of the System Architecture and its Platform Implementation. We provide feature comparison of DTaaS with other DT platforms in Platform Implementation. Afterwards, we illustrate the Implementation of Case Studies on the platform and describe the Performance Evaluation. The paper ends with Discussion and Future Work and Concluding Remarks.

Background and Related Work

Digital Twin Definition and Realisation

In abstract, DT is a digital representation of a physical object, that influences the future behaviour of the PT. However, this definition hides the variants and possible interpretations[12,13]. In this manuscript we refer to DT as a software running remotely in conjunction with cloud services that collectively add value to an existing PT through the use of models and simulations[14]. Some of the existing literature proposes three subcategories of DTs[15]: (i) *Digital Model*, in which physical object (PO) and digital object (DO) have no automated exchange of data, (ii) *Digital Shadow*, where PO emits data to the DO automatically and finally (iii) *Digital Twin*, in which case there exist a two-way automated data exchange between the DO and PO. Only in this case the PO is referred to as a PT. Nevertheless, a DT should be created to accomplish certain business goals, i.e., in terms of optimisation or provision of insights[16], and therefore, the inclusion of simulation models, inference and reasoning mechanisms, and data analysis become essential[17]. The orchestration for including these components towards achieving the business goals of DTs is called the *DT constellation*[18].

A five-level DT architecture has been proposed previously[19]. On the lowest level (i) Smart Connection, resides the data exchange between DT and PT; (ii) Data-to-information conversion, concerns conversion and aggregation of data for monitoring and to make it useful for (iii) Cyber, which is the central information hub and source of analysis across multiple data sources. At level (iv) Cognition, the knowledge acquired from lower levels are made available for decision-making, and finally at level (v) Configuration, is where decisions or reconfiguration from the DT is fed back to the PT, to make it self-adaptable. Thus, to create a DT, infrastructure, tools, models and configurations must address each level.

In the DT platforms discussed next, we focus on those that promote the implementation of the (iii) Cyber and higher layers, choosing to leave out platforms whose goal is to enable the Smart Connection and Data-to-information conversion layers, as platforms falling in this category have been surveyed before[17,20].

Re-usability in Digital Twin Implementations

Completely new development of a DT for each PT is not necessary if the DT or its parts are re-usable[21]. DTs should therefore be engineered and operated by reusing the parts previously developed for other DTs. Towards this end, we highlight two approaches, detailed next: asset reuse, and code generation.

Asset Reuse A prior work in the HUBCAP and DIGITbrain platforms[22] shows that a DT can be structured to be composed of re-usable assets other than models. The DIGITbrain project[23] introduces the notion of four re-usable assets: data, models[4], algorithms and behaviours. The software tools and frameworks have also been included in the algorithms asset. Each behaviour contains a pair of model and algorithm. The data and behaviours assets are selected and configured to create new DTs. Such re-usability can also be seen in both medical, manufacturing and maritime DTs[6,24–27]. The Digital Twin Platform Stack Architectural Framework of Digital Twin Consortium[28] proposes data, multiple representations of models (both static and computable), algorithms and services as re-usable assets.

Knowledge graphs[29] have been recognised early as a possible technology that can be used to configure DTs[30]. Subsequently, several DT architectures have used knowledge graphs. The SINDIT[31] is a component-based framework and architecture for rapid prototyping of DTs that uses knowledge graphs to represent the relationship between assets. A few cognitive twins[32] have used ontologies to connect simulation units[33] and component description[34].

Code Generation An alternative to re-usable assets, is the use of code generation techniques, informed by Domain Specific Languages (DSLs) descriptions, to generate DTs[21]. The proposed DSLs describe domain, data, tagging, constraint, and GUI as input to a set of generation tools, to create the application-independent parts of a DT platform. The proposed platform can then be tailored by a domain expert for a specific purpose, using a web-based front-end.

We argue that code generation approaches are complementary to asset reuse, since the configuration of such assets can be generated by means of a DSL. Several commercial

and open-source projects propose DSLs to specify composition of a DT as a configuration of models, relations and data exchange[35].

Microsoft Azure's Digital Twin Description Language (DTDL)[36] and Eclipse Vortolang[37] provide object-oriented modelling capabilities, such as *class*, *property*, *method* and *association*, as well at specifying data exchange endpoints[35]. The commercial platforms have come up with their own DT DSLs[5]. Examples include DTDL for Azure DTs, Eclipse Hono[38], Eclipse Ditto[39], and also AWS IoT twinmaker[40]. Other platforms, such as Eclipse BaSyX[41] are designed based on the Asset Administration Shell (AAS) meta-model, which is one of the outcomes of the German initiative Reference Architectural Model Industrie 4.0 (RAMI 4.0)[42], that recently turned into the IEC 63278-1 Standard[43]. Some of these platforms and frameworks have been previously analysed[17,44]. A feature comparison of these platforms and DTaaS is available in Comparison with Existing Platforms.

Comparison with Related Work

DTaaS takes a complementary approach across different DT frameworks and platforms. Multiple prior surveys[17,45,46] compare existing DT frameworks based on system architecture, interoperability, scalability, reproduction of previous results and composition. DTaaS intends to provide the infrastructure where complementary services can be orchestrated to run and maintain DTs with a well-defined DT asset configuration. Hence, there can be coexistence of existing DT platforms and DTaaS[47].

DTaaS provides two additional features not available in comparable DT platforms, namely, i) providing private workspaces for authoring and verification of re-usable assets and ii) enabling sharing of DT assets to foster collaboration and re-usability.

Some of the challenges in the theory-to-practice transition of DT platforms are the support for different operating phases of the DT and the orchestration of services[17,48]. DTaaS development process addresses these challenges by following successful software engineering practices such as microservices, DevOps and GitOps, which have found their way into the development and operation of DT platforms[49,50]. Additionally, it can include extended functionalities to perform verification of the published assets with DevOps[51] and provisioning of DT execution infrastructure by using Git workflows[52].

In DTaaS, the user is presented with two ways to work with a DT. In the simplest form a pre-existing, pre-packaged DT is made available on the platform. For example, a DT can be realised as a set of tightly-coupled services that exchange messages and stream data to/from a PT. Some of these services can execute models to provide their services. This form of DT is consistent with the DT implemented for an incubator machine[9,53]. In this case, the DT as a whole is re-usable. However, the services and the embedded models are tightly coupled preventing their reuse. The need for more enhanced re-usability motivates the second realisation of DTs which is the focus of this paper. In this paper, a DT is realised as a set of re-usable assets are categorised as *Data* assets, *Model*, *Function*, and *Tools*. A key innovation in our platform is that these four categories of assets, detailed later in Re-usable Digital Twin Assets, can be declared in the platform by the user, and made available to be reused by other users wishing to build DTs.

Monitoring of Digital Twins

Monitoring of DTs and there by their linked PTs is one of the use cases for DTs. Runtime monitoring tools are suitable for creating monitors especially from safety properties defined on either DTs or PTs. The Temporal Stream-based Specification Language (TeSSLa)[54] has been used to integrate monitoring into the described firefighter DT. TeSSLa refers to both a specification language and an associated tool-chain developed within a community-driven open-source initiative[55].

The fundamental principle of TeSSLa lies in its ability to describe transformations of data streams from inputs to outputs. It can be used within DTs to monitor (i) data from PT and (ii) predictions of DT. In both cases, the deviations can be reported to either users or to decision making software.

TeSSLa specifications are constituted by a set of input stream declarations. The specification defines derived streams created by applying specific operators to the input and previously defined streams. Some of these derived streams are designated as the specification's output streams. TeSSLa provides libraries and supports the creation of macros. Macros are user-defined stream operations constructed from the core operations or other macros, thereby aiding the specification of complex properties. TeSSLa specifications allow for the creation of advanced output streams, including statistical data, with timestamps for events and support compiling into monitors binary, executable monitors that may be integrated into DTs.

Case Studies

Incubator

The incubator PT is a box containing a heater and a fan connected to a controller that can turn on and off the heater/fan. The controller is supported by temperature sensors placed inside the box. The incubator has a single requirement which is to save energy if the user forgets to close the lid, or simply misplaces it. The incubator PT is used to produce tempeh[56], which is a food produced by growing mold around soybeans. The original method for producing tempeh, discovered in Indonesia, consists of: 1. **Hydration of soybeans:** Soybeans are soaked either overnight in cool water or for a shorter period in hot water. 2. **Removal of skins:** The hydrated beans are rubbed together with water, either before or after cooking, to remove skins. The loosened skins float away with the water. 3. **Partial cooking:** The beans are partially cooked by steaming or boiling, retaining firmness. 4. **Inoculation:** After cooking, all liquid is drained, and the beans are inoculated with scrapings from a previous fermentation. 5. **Fermentation:** The inoculated beans are wrapped in banana leaves and placed in a warm location to ferment. Mold mycelia bind the beans into a compact cake, and enzymic digestion softens individual beans. 6. **Further processing:** Raw tempeh can be sliced and dried, roasted, cooked in soup, or deep-fried before consumption.
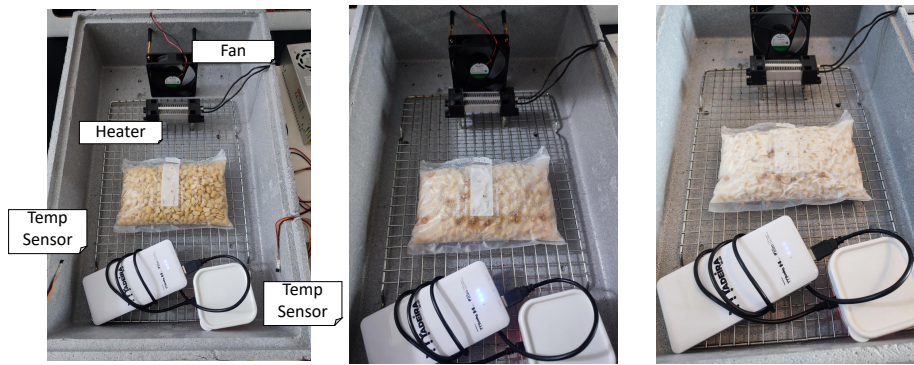
Figure 1. Example fermentation of tempeh using the incubator PT at the beginning (left), halfway through (middle), and finished (right).

The incubator is designed to support the fermentation process, where the user is expected to open the box lid to inspect its contents and determine whether the fermentation has been concluded. Figure 1 shows an example tempeh production process at the beginning (left), halfway through (middle), and finished (right). As can be seen, the mold mycelia in white grows around the soybeans, consuming them and becoming a compact cake. What makes this case study deceivingly simple is the fact that at its highest growth rate the tempeh switches from being a heat sink into a heat source[56], effectively changing the dynamics of the system. This effect varies substantially from batch to batch because it is a biological system and depends on the number of factors such as the amount of starter mold as well as its genetics. Detecting the open lid is just a simplified use case of the digital twin used in this paper to illustrate the DTaaS platform.

Firefighter

The firefighting scenario, part of the O5G-N-IoT project[10], aims to enhance decision-making and operational safety in emergency response situations. Firefighters equipped with self-contained breathing apparatuses (SCBAs), including oxygen tanks, operate in hazardous, rapidly changing environments. This scenario presents various challenges that necessitate quick, informed decisions based on accurate, real-time information.

Current firefighting practices involve firefighters periodically reporting their oxygen tank pressure to a mission commander. This method suffers from delays in information updates and the potential for human error, hindering real-time situational awareness. In contrast, a firefighter DT offers continuous, real-time monitoring and automatically updating crucial data like oxygen levels and firefighter locations. Using a firefighter DT significantly improves response strategies, adapting quickly to changes in the operation.

The primary requirements[57] for the firefighter DT are continuous monitoring of firefighters' oxygen levels and locations within the building, providing actionable insights to the mission commander based on real-time data, and ensuring the safety of firefighters by predicting oxygen depletion and recommending safe exit routes. The primary challenge is ensuring firefighter safety while maximising operational efficiency, which involves continuously monitoring SCBA oxygen levels and

navigating through the building using real-time data. The firefighter DT and the coupled PT can also be used to simulate emergency missions.

The firefighter DT described in this paper requires the existence of localisation services and the continued validity of the building model. The building under fire likely undergoes structural deterioration and transformation. If the changes in the building significantly deviate from the building model used in the firefighter DT, then the predictions made by the DT become invalid. Another limitation could be the over-reliance on a robust localisation service in a very dynamic environment (building under fire). The current version of firefighter DT is not capable of handling intermittency in localisation service. Further explanation on the limitations of the firefighter DT in terms of data collection, necessary prerequisites and scope are available in a previous publication[57].
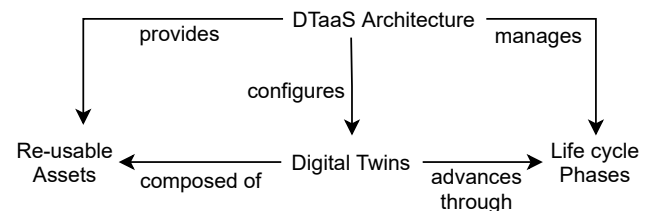

Figure 2. Relations between central concepts.

Re-usable Digital Twin Assets

In the following sections we introduce the core notions needed for an architecture for a DT platform that facilitates reuse of different models, tools and other heterogeneous components:

- The Digital Twin assets, which represent the smallest elements of a DT that can be reused

- the Digital Twin Definition itself, and its composition from DT assets and services as *DT configuration*,

- the Phases in Digital Twin Life Cycle, and their relation to DTs, and finally

- the System Architecture that connects all these.

The relation between these concepts is shown in Figure 2.

Let us turn to re-usable DT assets first. DTaaS treats DTs as having re-usable assets. In short, a DT asset is the smallest element of a DT that can be reused. These assets are put together and configured in a certain way. We use four categories of assets: data (D), model (M), function (F) and tool (T). This section provides explanations for categorisation of DT assets.

The data (D) asset refers to data sources and/or sinks available to a DT. Typical examples of data sources are sensor measurements from the PT, and test data provided by manufacturers for calibration of models. Typical examples of data sinks are visualisation software, external user software and data storage services. There exist special outputs such as events, and commands which are akin to control outputs from a DT. These control outputs usually go to the PT, but they can also go to another DT[58].

The model (M) assets are used to describe different aspects of a PT and its environment, at different levels of abstraction. Therefore, it is possible to have multiple models for the same PT. For example, a flexible robot used in a car production plant may have structural model(s) which will be useful in tracking the wear and tear of parts. It can have a behavioural model(s) describing the safety guarantees provided by the robot manufacturer. It can also have a functional model(s) describing the part manufacturing capabilities of the robot. Models have inputs, outputs, states, and parameters, as well as initial values for states, following the nomenclature of the Functional Mockup Interface[59] standard.

The function (F) assets are primarily responsible for pre- and post-processing of data. The data to DT comes from either PT or databases. This data is used for evaluation of models. Given that a DT might have multiple models, the data (and its format) required is not going to be the same for all of them. This observation is acutely true in cases of models developed long after data has been captured. Another case is that of normalisation and unit conversion of data; well-implemented pre-processing functions help with this task. In the same vein, post-processing functions help with the conversion of model outputs into valid control inputs to PT. One such example is that of controlling a machine speed based on predictions of a machine learning model. The model provides normalised (say zero to one) prediction which must be converted into actuator input for the machine.

The software tools and frameworks (in short, tools) contain implementations of engineering domain-specific or generic computer algorithms. A few examples of domain-specific algorithms are: SIMPLE (computational fluid dynamics), Barycentric method (graph drawing), SPICE and Xyce (electronic circuit simulation), ant colony optimisation (genetics), and generative adversarial networks (machine learning). A few examples of generic algorithms are: Newtons method (numerical), Bubble sort and Strassen matrix multiplication. Domain-specific tools such as Maestro CoE[60] contains implementation algorithms for conducting co-simulation. Same is the case with OpenFOAM[61] and Tensorflow[62]. On the other hand, Matlab[63] is an example of a tool that has implementations of algorithms coming from many engineering domains.

These tools are executed on top of a computing platform, i.e., an operating system, or virtual machines like Java Virtual Machine, or inside Docker containers. The tool are used to create, evaluate and analyse models. Most models require tools to evaluate them in the context of data inputs. There exist cases where models and tools are combined. The combined entity is run in the same way a tool is run.

Incubator

Let us now investigate the assets used in the incubator case study. Figure 3 provides an overview, and we detail the different kinds of assets next.

The incubator DT therefore has the single requirement: to save energy if the user forgets to close the lid, or simply misplaces it. This requirement is implemented by the following DT services:

- *Temperature Prediction* – Responsible for predicting the average temperature in the box at time t+H from the average temperature, control actuation, and room temperature, at time t. This service is triggered whenever a new control actuation is issued.

- *Lid Open/Closed Detection* – Detect and signals, whether the incubator lid has been opened or closed.

The above services work together to fulfil the energy saving requirement as follows:

1. The open lid detection service compares the temperature predictions with the actual temperature, and when there is a deviation it signals the lid as being open.

2. The supervisor then re-configures the controller accordingly to save energy by placing it in a low power mode.

We refrain from giving more details about how these different services coordinate, which is explored in prior work[9], and focus on the assets used in the incubator DT.

The data assets are an InfluxDB time-series database and a RabbitMQ broker. The InfluxDB time-series database acts both as data store and visualisation service. The RabbitMQ endpoint acts as bidirectional communication broker between PT and DT.

An example function is the calculation of the average between the two temperature sensors installed inside the box. If analogue temperature sensors were to be used, one would also need smoothing functions to eliminate the noise from the temperature measurements.

The models are:

Plant Models The incubator case study uses four different models for the plant itself, each for a different aspect.

- *2-Parameter Model*: A simple scalar ordinary differential equation (ODE) representing the plant dynamics. It takes as inputs the room temperature, the state of the heater, and returns as an output the average temperature of the air volume inside the box. It contains two parameters related to the box-walls heat conductivity and air heat-capacity.

- *4-Parameter Model*: A plant dynamic model (ODE) that takes as inputs the room temperature,
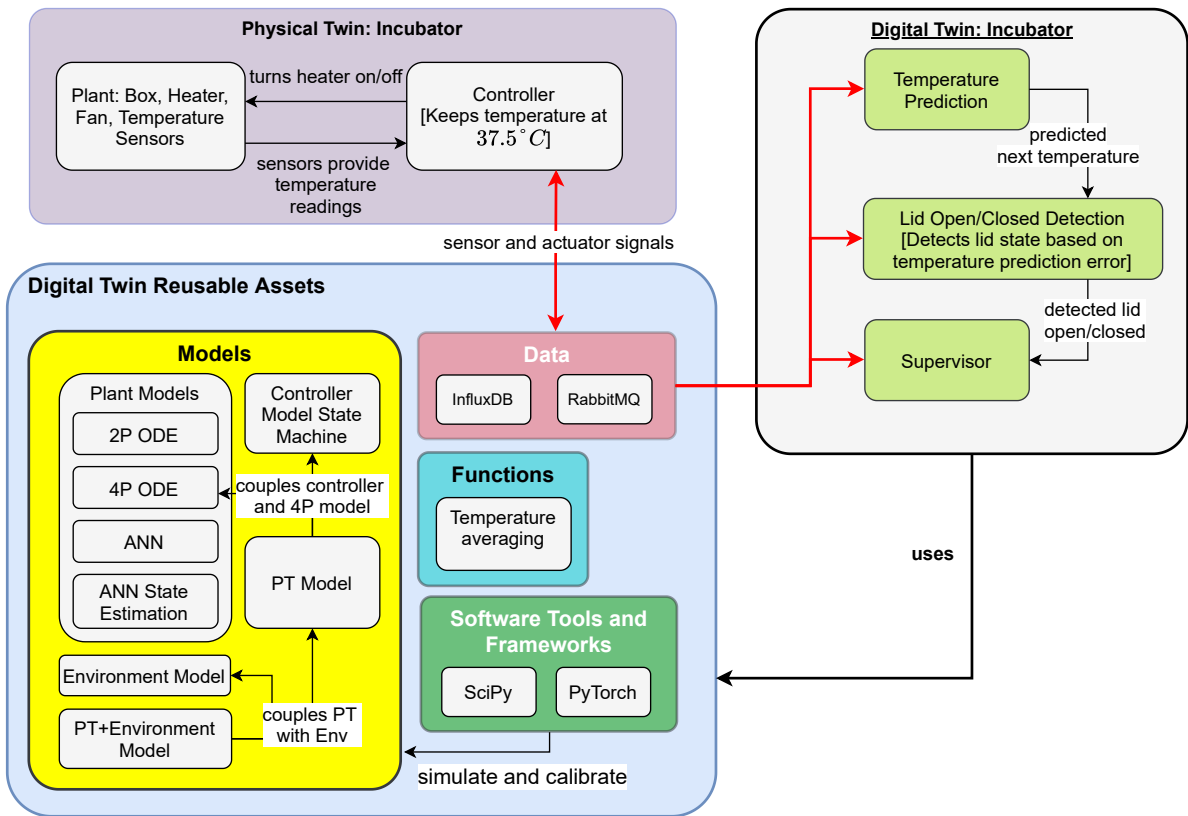
Figure 3. Summary of the assets used in the incubator running example.

the state of the heater, and returns as an output the average temperature of the air volume inside the box, as well as the temperature of the heating element.

- *ANN Model*: A neural network based model, that takes as inputs the room temperature, the state of the heater, the previous average temperature of the air volume inside the box, and returns as output the average temperature of the air volume inside the box.

- *ANN State Estimation Model*: A neural network based model, that takes as inputs the room temperature, the state of the heater, the previous average temperature of the air volume inside the box, and returns as output the average temperature of the air volume inside the box as well as the temperature of the heating element.

Controller Model A controller model (state machine), that takes as input the average temperature of the air volume and outputs the state of the heater.

PT Model A hybrid automata model. The input is the room temperature, and the set of parameters is the union of the 4-Parameter model and the controller state machine model.

Environment Model A simple algebraic model $T_r = A\sin(Bt) + C$ representing the temperature of the room as a function of the time (in seconds). The parameters are A, B, C.

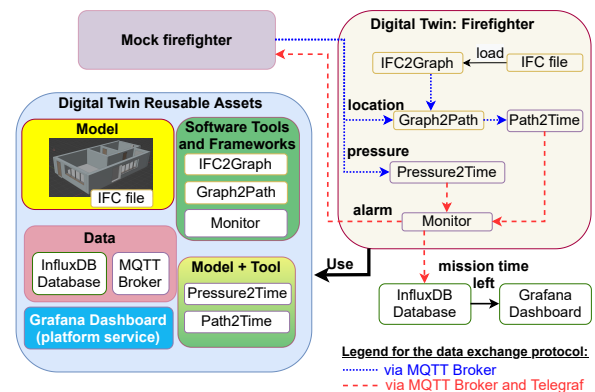**PT + Environment Model** A model coupling the PT and Environment models.

Figure 4. Re-usable assets included in the firefighter DT.

Finally, tools such as the numerical solver from SciPy[64], and the neural network framework PyTorch[65] are used.

Firefighter

The firefighter DT continuously monitors self-contained breathing apparatuses (SCBA) pressure levels and firefighter locations. If the oxygen level is deemed too low in relation to the fastest escape route, the DT alerts the individual firefighter. It also recommends the quickest and safest exit route based on the current location of the firefighter and building layout. DTaaS user in this scenario is a firefighter sub-contractor or mission controller, who deploys and adapts the system before firefighting missions take place.

Each firefighter DT is realised using the following assets. The Data used in this DT is provided by a mock PT (mock firefighter). Simulated SCBA pressure data and location data
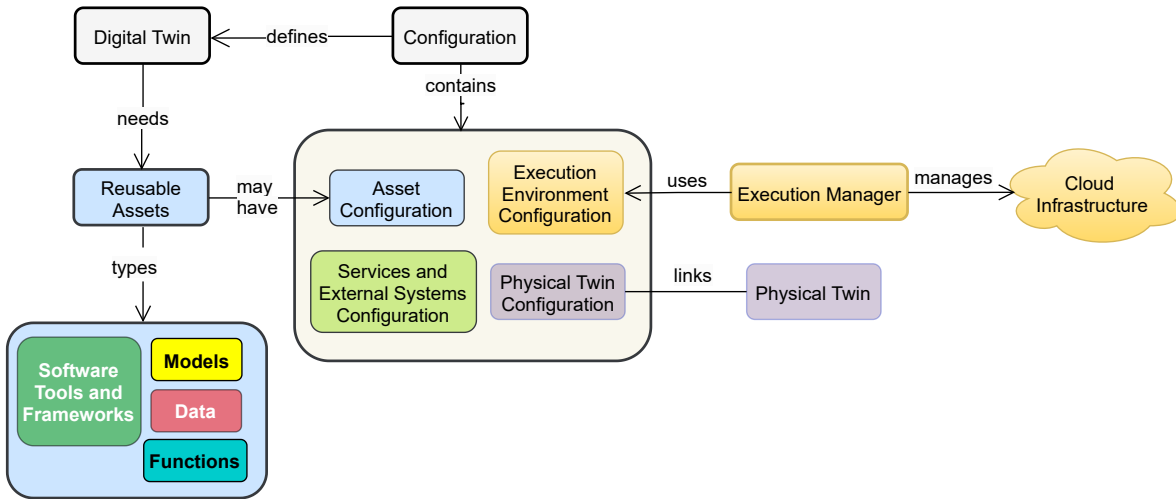
Figure 5. A conceptual relationship between DT assets (Data, Model, Function, Tool) and DT configuration[66].

from inside of the building are sent to a MQTT broker to be ultimately used by the DT. The communication between PT and DT is based entirely on MQTT topics. Thus the DT case study outlined in this paper works for a real firefighter PT as long as the location and oxygen level data is sent to the DT. It then produces a prediction on mission time and raises an alarm if the oxygen level is deemed too low. The remaining mission time is also stored in InfluxDB – a time-series database. The InfluxDB acts as data storage. Real-time data visualisation using InfluxDB and Grafana dashboards displays crucial information such as oxygen levels and remaining mission time for the usage by the mission controller.

The 3D-building data specified as an Industry Foundation Classes (IFC)-file serves the role of model to the DT. Three tools are included in this DT.

- *Building Data Transformation* a tool that transforms building data from the IFC-file into a graph structure for navigation purposes. This tool is essential for creating an accurate and navigable representation of the building within the DT.

- *Route Determination* determines the shortest path using the graph derived from building data. This tool plays a key role in the continuous route planning of the system.

- *Monitor* is a run-time monitor for maintaining the safety of a firefighter. This monitor sends an alert if there is insufficient oxygen for the firefighter to safely exit a firefighting situation.

In certain cases, the models are integrated tightly into the tools and are published as one re-usable asset. Such integrations can be considered as model-tool pairs. There are two tools with integrated human physiological models in the firefighter DT. They are:

- *Air Consumption Calculation* is a tool for calculating the shortest path information to calculate the amount of air consumed during navigation. This tool is vital for approximating the rate of oxygen depletion and managing the firefighters' air supply.

- *Pressure to Time* uses the pressure sensor values to approximate the amount of time the air in the SCBA's air cylinder will last.

Every asset in this list is reused for serving multiple firefighters with one DT for each firefighter. Multiple firefighters DTs can coexist at the same time on the platform and their collected data can be shown in the same dashboard.

Digital Twin Definition

Having defined the DT asset, we now turn to their composition into DTs. Indeed, for our purpose a DT is mainly a (constrained) set of assets, called a DT configuration, connected to a physical system - the PT - using some services.

First, we investigate the constraints in the relation between assets to have a valid DT configuration. There is a dependency between the assets especially in the context of creating DTs (Figure 5): Only functions/tools can use models/data. A specific combination of these assets constitute a DT. We distinguish between the *digital twin design* (a set of assets) and its *digital twin configuration*.

Digital Twin Design A DT design D_t describes the possible sets of assets that a concrete DT instance can have. Here, $\mathcal{P}(\cdot)$ denotes the power set and $\mathcal{P}_>(\cdot) = \mathcal{P}(\cdot) \setminus \{\emptyset\}$. The possibilities of asset combinations used in a DT are expressed below (Definition 1), where a DT design D_t is a tuple of the possible data assets (denoted D), model assets (denoted M), function assets (denoted F) and tool assets (denoted T).

$$D_t \in \mathcal{P}(D) \times \mathcal{P}(M) \times \mathcal{P}(F) \times \mathcal{P}_>(T) \qquad (1)$$

We demand that a digital twin design has at least one tool.

Digital Twin Configuration The interconnections between assets, parameters (configurable run-time variables) of the assets need to be specified for each DT. This information becomes a part of the DT asset configuration (C_a). The information encoded in C_a is not sufficient to truly manifest a closed-loop communication between a DT and a PT. Thus, each DT requires a complete configuration (C_{dt}) that is
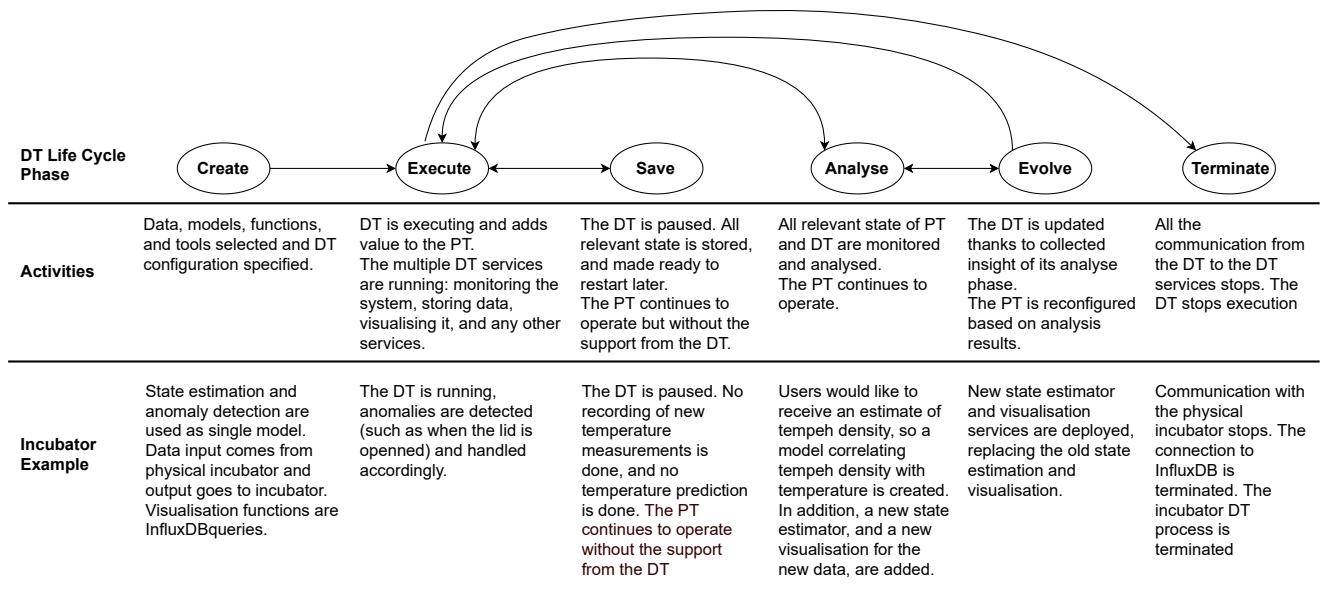
DT Life Cycle Phase	Create	Execute	Save	Analyse	Evolve	Terminate
Activities	Data, models, functions, and tools selected and DT configuration specified.	DT is executing and adds value to the PT. The multiple DT services are running: monitoring the system, storing data, visualising it, and any other services.	The DT is paused. All relevant state is stored, and made ready to restart later. The PT continues to operate but without the support from the DT.	All relevant state of PT and DT are monitored and analysed. The PT continues to operate.	The DT is updated thanks to collected insight of its analyse phase. The PT is reconfigured based on analysis results.	All the communication from the DT to the DT services stops. The DT stops execution
Incubator Example	State estimation and anomaly detection are used as single model. Data input comes from physical incubator and output goes to incubator. Visualisation functions are InfluxDBqueries.	The DT is running, anomalies are detected (such as when the lid is openned) and handled accordingly.	The DT is paused. No recording of new temperature measurements is done, and no temperature prediction is done. The PT continues to operate without the support from the DT	Users would like to receive an estimate of tempeh density, so a model correlating tempeh density with temperature is created. In addition, a new state estimator, and a new visualisation for the new data, are added.	New state estimator and visualisation services are deployed, replacing the old state estimation and visualisation.	Communication with the physical incubator stops. The connection to InfluxDB is terminated. The incubator DT process is terminated

Figure 6. Mapping of the DT life cycle phases to the incubator use case[67]. The ODE acronym stands for ordinary differential equation.

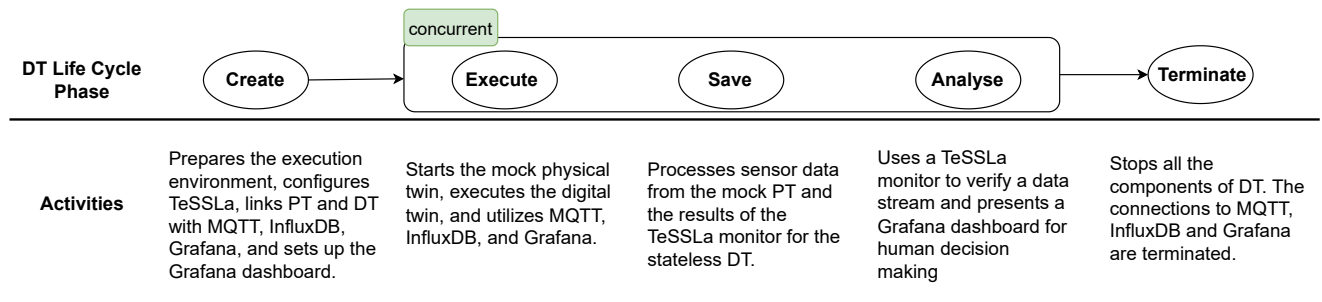| DT Life Cycle Phase	Create	Execute	Save	Analyse	Terminate
Activities	Prepares the execution environment, configures TeSSLa, links PT and DT with MQTT, InfluxDB, Grafana, and sets up the Grafana dashboard.	Starts the mock physical twin, executes the digital twin, and utilizes MQTT, InfluxDB, and Grafana.	Processes sensor data from the mock PT and the results of the TeSSLa monitor for the stateless DT.	Uses a TeSSLa monitor to verify a data stream and presents a Grafana dashboard for human decision making	Stops all the components of DT. The connections to MQTT, InfluxDB and Grafana are terminated.

Figure 7. Mapping of the DT life cycle phases to the firefighter use case.[57]

sufficient to execute a DT in the presence of supporting services and execution environment.

The C_{pt} denotes configuration information required by a DT to communicate with a PT. Each DT may have constraints on the kind of execution environments it is capable of using, i.e. tools that can only run either on a specific operating system or on a server with specific hardware capabilities. The C_i denotes the infrastructure configuration required by a DT. The C_e denotes configuration for integration of a DT with external software systems and services, e.g. third-party visual dashboards.

$$C_{dt} \in C_a \times C_i \times C_e \times C_{pt} \qquad (2)$$

Among all the configurations shown in Definition 2, C_a and C_{pt} are very specific to one DT design D_t. Thus, generalisation of these two configurations into a configuration specification standard is a challenging task. The other two configurations – C_i and C_e – are more general and a configuration specification standard for these two is a manageable challenge. A sanity check is required on validity of any given C_{dt}.

We denote an instance that stems from a design D_t configured with a configuration C_{dt} with D_tC_{dt}.

Table 1. DT configuration for the incubator and firefighter case studies.

Configuration for	Case Studies	
	Incubator	Firefighter
Assets (C_a)	asset selection; configuration for models, ex: 4-parameter model	asset selection
Infrastructure (C_i)	docker	docker
Services (C_e)	RabbitMQ, InfluxDB	MQTT, InfluxDB, Grafana
Physical Twin (C_{pt})	Plant controller parameters	Not Applicable

Three situations demand adjustments to C_{dt}. First is a user-driven change in C_a, C_{pt}, C_i, C_e, or C_{dt} of included DTs. In this case, a validity check is required before a transition to new a configuration can be made. Second is

a requirement to perform what-if analysis. What-if analysis requires minor variations on C_{dt} to plan and optimise future steps to be undertaken either on a PT or a DT. Actual implementation of what-if analysis can be resource intensive with the resource requirements scaling up in proportion to algorithmic bounds on the (sub)-systems being used by a DT. Third is the reuse of a DT by one or more users. Reuse at minimum requires reconfiguration of credentials to external services. In addition, users might want to change parameters in C_a before using a DT.

A DT can also use external tools such as planning and optimisation. This is especially true in what-if analysis. If these tools are used exclusively within a DT, then, they can be considered as tools in the asset library. Otherwise, they are part of the infrastructure / external world.

Case Studies A mapping of C_{dt} to the incubator and firefighter case studies is shown in Table 1. There are many models suitable for use in incubator DT and some of these are parameterised models. The model selection and their configuration is expressed in C_a. The models and tools used for firefighter are expressed in C_a. Both these case studies run inside docker containers. Each of them use platform services for communication with real/mock PTs. The incubator uses RabbitMQ for PT-DT bidirectional communication. Both the case studies use InfluxDB for storing the PT data and analysis results. In addition, the firefighter case study uses MQTT for data transmission from PT to DT and Grafana for visualisation.

Phases in Digital Twin Life Cycle

A digital twin is a not a static structure – it transitions through a life cycle with different phases, and we consider it to be one of the main tasks of a DT platform to manage DTs in all these phases, as well as the transitions between these phases.

Two pioneering standards for DTs, namely, the ISO 23247 Standard [68] (*Digital Twin framework for manufacturing*) and the IEC 63278 Standard [43] (*Asset Administration Shell for industrial applications* which is based on the RAMI 4.0 model), use life cycle as a general principle for defining DTs. The former motivates the DT life cycle from the definition of product life cycle. However, such a definition states that DTs are more useful if the DTs in effect can be shared across the product life cycle, including *design*, *planning*, *production*, *maintenance*, and *support*. Similarly, the IEC 63278 standard motivates DT life cycle based on the life cycle of the physical asset that the digital representation is featuring, which can be done in a product or process. Digital twinning a product or a process require different (yet similar) life cycle phases, such as *design*, *manufacturing*, *usage/maintenance*, and *end of life treatment* for products, and *design*, *implementation*,*operation/maintenance*, and *decommission* for process. The life cycle phases so far are for DTs themselves. There is a need to translate these product life cycle phases onto DT platforms without the loss of conceptual and practical nuances. The rest of this section describes DT life cycle phases on DT platforms. The mapping of these phases to product life cycle phases is explained at the end of the section.

A DT life cycle on DT platforms consists of the *create*, *execute*, *analyse*, *evolve*, *save*, and *terminate* phases. The platform allows a user either to create a new DT or select an existing DT to engage with it in any of its phases. Thus, reuse is possible by reusing assets that existed before the DT is created, or reusing assets from prior runs and phases of the same DT. We next describe the phases on a conceptual level, and describe the used techniques in the case studies to illustrate them.

Create and Execute Phases The *create* phase involves asset selection and specifying DT configuration. There is no creation phase at the time of DT reuse. The *execute* phase involves an automated execution of a DT based on its configuration.

Analyse and Evolve Phases The *analyse* and *evolve* phases are concerned with monitoring and potential reconfiguration of a DT. Monitoring, at its most basic level, requires data gathering and storage, of the interaction between the PT and its environment, and among the PT's assets. However, even for a simple system, such as the incubator[67], monitoring requires that hidden quantities (that is, quantities for which we cannot obtain a sensor measurement directly) be estimated. This activity corresponds to *analyse* life cycle phase. Often these hidden quantities are represented by variables in various models used by the DT. The consequence is that estimates of these hidden quantities need to be stored in database, becoming then input to decision-making simulations, where all variables of the models need to be properly initialised. For more details we refer the reader prior work[53].

Monitoring also informs the next life cycle stage of DT: the *evolve* stage. The evolve phase involves user/event-triggered reconfiguration of an instantiated DT. Note that the monitoring and planning steps make use of the other DT services.

Reconfiguration and Consistency The evolve phase requires reconfiguration of DTs. The aim of reconfiguration is to ensure *consistency* between DT and its PT, i.e., the adequacy of the DT to mirror its PT, access its data, and enable the required analyses.

Reconfiguration may be triggered by different kinds of events, three of which we discussed above, they are specific to the system and, thus, reconfiguration procedures must be provided by the user. These procedures are highly application specific. The reconfiguration procedures must be able to access the current DT configuration and its assets. As the configuration is highly heterogeneous, the platform should offer a way for *uniform* access to it, i.e., a representation mapping μ that is defined on C_{dt} *and all its assets* as well as provide an interface for the user to program re-configurations in terms of the uniform access, i.e., define transitions $\mu\left(C_{dt}^l\right) \to_\mu \mu\left(C_{dt}^{l+1}\right)$. Knowledge graphs are a suitable technology to implement such mappings. The approaches to express consistency between DT and PT in terms of queries on knowledge graphs have been shown to be useful[69].

Save and Terminate Phases The *save* phase involves saving the state of DT to enable future recovery. If a DT is reused, there will be a temporal gap between creation and execution times of a DT. Thus, a need might arise for just-in-time DT reconfiguration at the point of execution.

The *terminate* phase involves stopping the execution of the DT and releasing all the resources and connections mentioned in the DT configuration. It is also responsible for cleanup and correct shutdown of the PT (via its components), e.g., for hardware-in-the-loop testing.

The create life cycle phase of DTaaS corresponds to creation of first prototype DT perhaps during the design phase. This create phase also corresponds to preparation of DT platform environment for DT execution. This DT will have to be executed to glean insights; such DT execution corresponds to execute life cycle phase via dedicated life cycle script / program. Depending on the results of execute life cycle phase, transition to other phases can happen. The save life cycle phase can be used to save DTs for reuse, i.e. future execution. The analyse and evolve phases together can help with DT management during the complete PT product life cycle. The terminate phase corresponds to termination of execution for a DT. This phase is specific to DT platforms and does not correspond to any PT life cycle phase.

Incubator

Figure 6 contains a mapping of different activities done for the incubator DT and the conceptual description of DT life cycle phases.

In the create phase, state estimation and anomaly detection are combined into a single model. Data input from the physical incubator is processed, and the output is sent back to the incubator. Visualisation functions are handled using InfluxDB queries.

During the execute phase, the Digital Twin (DT) runs, detecting and addressing anomalies, such as when the incubator lid is opened.

Transitioning to the save phase, the DT is paused. No new temperature measurements are recorded or predicted, but the physical twin (PT) continues to function independently of the DT.

In the analyse phase, users require an estimate of tempeh density. A model correlating tempeh density with temperature is created, along with a new state estimator and visualisation for the new data.

In the evolve phase, new state estimation and visualisation services are deployed, replacing the previous versions.

Finally, in the terminate phase, communication with the physical incubator ceases, the connection to InfluxDB is terminated, and the incubator DT process is concluded.

Firefighter

DTaaS platform's management of firefighter DT is facilitated through a series of life cycle phases, detailed in the overview graphic in Figure 7.

The DT assets are configured in *create* phase to create firefighter DT and to prepare the execution environment. The TeSSLa is configured, links between PT and DT are created using MQTT, InfluxDB, and Grafana dashboard is setup. *Execute* phase activates mock PT and DT. DT now employs MQTT for data transmission, InfluxDB for storing data, Grafana for displaying real-time information. *Save* and *analyse* phases are run concurrently to execute phase. The system's design ensures that the most recent data from mock PT is always available, with historical data primarily used

for human decision-making. In *analyse* phase, the TeSSLa monitor continuously checks for oxygen levels and sends an alert when the level is too low. In essence, the possibilities for firefighters to escape is continuously analysed using monitor. The results are streamed in real-time to Grafana dashboard. This concurrent activation of execution, save and analysis phases makes DT responsive and effective. The process concludes with *terminate* phase in which all the executing DT assets are terminated, connections to MQTT, InfluxDB, and Grafana are closed, ensuring an orderly shutdown of DT operation.

Through these life cycle phases, DTaaS manages the complex operations involved in the firefighting scenario, ensuring efficient data handling and real-time analysis for decision-making.

System Architecture

Having established the notions of DT assets, DT configuration, and DT life cycle phases, we now examine the requirements for a platform that uses these concepts to manage DTs, as well as describe the microservice-based software architecture underlying DTaaS that realises these requirements.

Requirements

The HUBCAP project[70] promoted the collaborative model-based engineering. One of the outcome of the project is a collaborative sandbox platform in which the providers of the model authoring tools save their tools pre-packaged in virtual machines (VMs). The models are published as either public or private packages inside their workspaces. The users of tools instantiate VMs and work with their models. This experience has brought forth the need to support on-platform creation of DT assets. With the advances in virtualisation, it is now possible to support software tools inside containerised environments (Docker, Kubernetes etc.). In addition, there are cases where the generation of DT assets can be done impromptu. For instance, the safety monitoring use cases use runtime monitors like NuRV[71] which can generate the required model(s) for just-in-time execution of DTs.

The large-scale collaboration among partners creating DTs requires the ability to share re-usable assets and then help users discover the availability of existing assets. For example, the co-simulation based DTs reuse Maestro tool. In addition, there is also significant reuse of functional mockup units (FMUs). Some of these FMUs are confidential. So there is a need to facilitate both the private and common spaces for DT assets. The DIGITbrain project[23] promoted the use of structured metadata for facilitating the discoverability of DT assets. It is important to note that the consolidation here refers more towards discoverability and less towards centralisation. As long as the assets are discoverable and integrable into DTs at execution time, their storage location is irrelevant. Such a decentralised storage of DT assets respects commercial advantages of asset owners and is not an impediment to DT execution[72].

The value of DTs is realised at execution time. This execution can either be manual or automated. It can also happen in centralised or distributed fashion. All these are possibilities DTs may take advantage of. For instance,
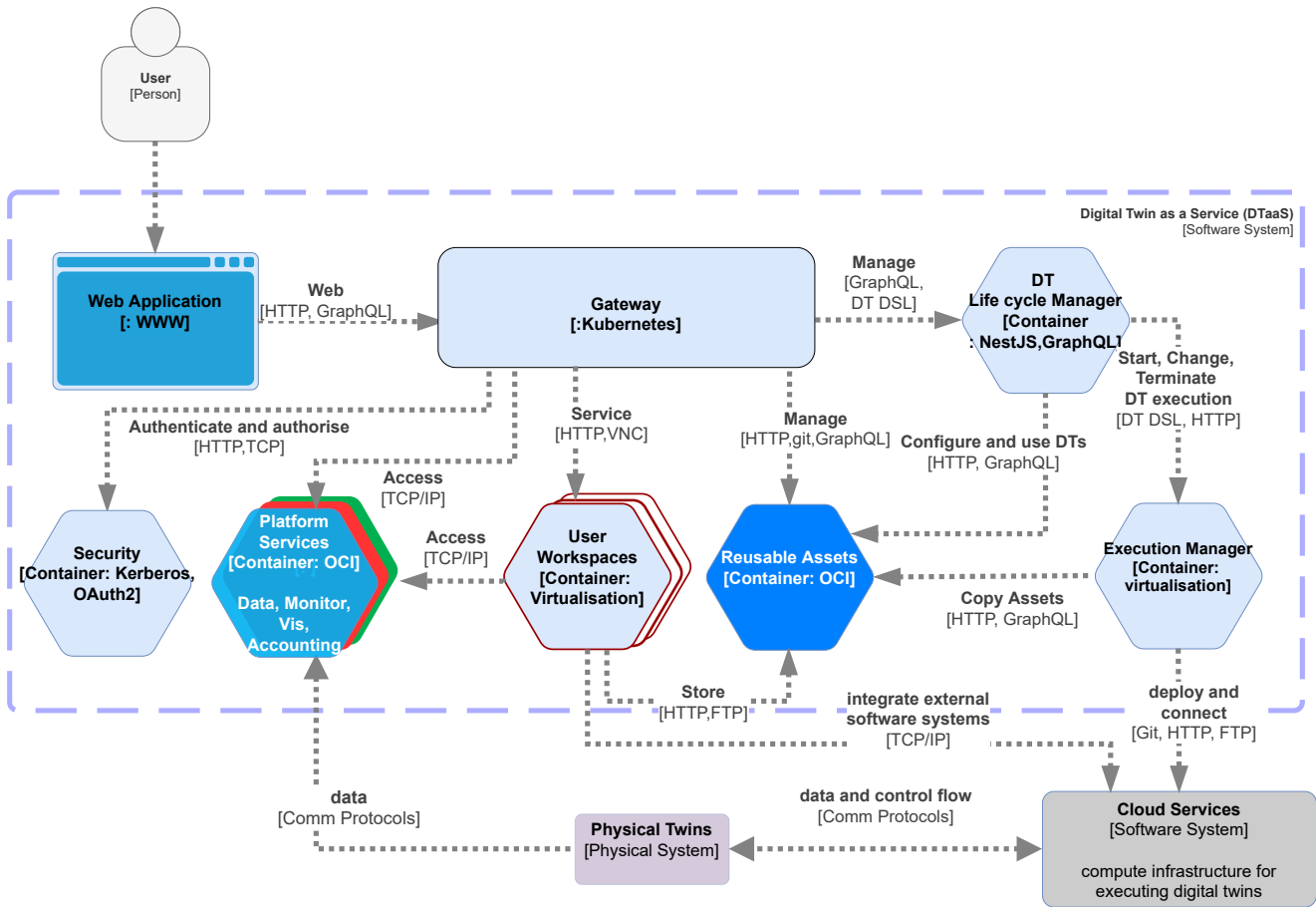
Figure 8. The software architecture of DTaaS platform. It is based on microservices architecture.

incubator DT can be executed by users in their workspaces, or it can be executed as an automated service. It is also possible to package incubator DT into Helm Chart and execute the same on a Kubernetes cluster. Thus DT platforms need to support diverse execution environments for DTs.

The previous sections explain the advantage of re-usable DT assets and creation of DT from these assets using configuration. Thus support for configuration and reconfiguration is a vital feature on any DT platform. Some widely used DT platforms like Azure DT, Eclipse Ditto promote configurability as a means of managing DTs.

Monitoring and predictive maintenance are two prominent use cases for DTs. Monitoring of civil infrastructure such as bridges and power infrastructure such as wind turbine farms require DTs which are either 24x7 available or executable on-demand (CP-SENS project[73]). For instance, a safety monitoring team might perform audit on the wind turbine farm and schedule the next audit after 12 months. If it is not desirable for the DT (as relevant to the audit scenario) to be run in the interim it is advantageous to save and re-instantiate the DT at the time of next audit. One of the key activities of a safety audit is to perform what-if scenario analysis to gauge the safety of the wind turbine farm under alternative weather conditions. The DT platforms must facilitate such a use case. One of the exemplar case studies developed on DTaaS is a fault injection based co-simulation. In this use case, faults are provided as DT configuration for a what-if scenario analysis.

Finally, users do have either direct or indirect interaction with DTs. The direct interaction needs to be facilitated

via DT platform while the indirect interaction happens via services running on top of DTs. For example, the incubator DT integrates with InfluxDB on which pre-configured dashboards are loaded. The users can interact with these dashboards instead of directly interacting with incubator DT. Summarised. the discussed user requirements are:

Requirement-1: Author – create different assets of the DT on the platform itself. This step requires use of tools whose sole purpose is to author DT assets.

Requirement-2: Consolidate – consolidate the list of available DT assets and authoring tools so that user can navigate the library of re-usable assets. This functionality requires support for discovery of available assets.

Requirement-3: Configure – support selection and configuration of DTs. This functionality also requires support for validation of a given configuration.

Requirement-4: Execute – provision computing infrastructure on demand to support execution of DT.

Requirement-5: Explore – interact with DT and explore the results stored both inside and outside the platform. Exploration may lead to analytical insights.

Requirement-6: Save – save the state of DT that is already in the execution phase. This functionality is required for on-demand saving and re-spawning of DTs.

Requirement-7: What-if analysis – explore alternative scenarios to (i) plan for an optimal next step, (ii) re-calibrate new DT assets, (iii) automated creation of new DTs or their assets; these newly created DT assets may be used to perform scientifically valid experiments.

Requirement-8: Collaborate – share DT assets with other users of their organisation. If DTs are to be confidential, services built on DTs can be shared.

System Components

Despite diverse requirements for different users, DTaaS presents a unified interface to the users. This unification is achieved by providing a gateway to consolidate the functionality provided by internal system components. Figure 8 shows the C4 system architecture[74] of DTaaS software platform. Users interact with the software via a web application. The gateway is a single point of entry for direct access to the platform services and is responsible for controlling user access to the containers. Each container contains a set of microservices. The term microservices shall be used henceforth to explain the system functionality implemented in C4 containers. The swapping of terms is to align with the architectural description of DT platforms in the existing literature[75]. The microservices are complementary and composable; they fulfil core requirements of the system. There are microservices for catering to store, explore, configure, execute and scenario analysis requirements. Dedicated user workspaces enable authoring of DT assets. The microservices are packaged and deployed on Kubernetes cluster to take advantage of virtualisation, scalability, and service discovery.

Microservices

The microservices illustrated in Figure 8 provide the bulk of the platform functionality. The security microservice implements role-based access control (RBAC) in the platform. The accounting microservice is responsible for keeping track of the platform, DT asset and infrastructure usage. Any licensing, usage restrictions need to be enforced by the accounting microservice. Accounting is a prerequisite to commercialisation of the platform. Due to significant use of external infrastructure and resources via the platform, the accounting microservice needs to interface with accounting systems of the external services.

The data microservice is a front-end to all the databases integrated into the platform. A time-series database and a graph database are essential, from multiple reported cases in the state of the art[77]. These two databases store time-series data from PT, events on PT/DT, commands sent by DT to PT. It is possible to integrate dedicated Internet of Things (IoT) frameworks such as Eclipse Hono[38] into the data microservices. The PTs uses these databases even when their respective DTs are not in the execute phase.

The visualisation microservice is again a front-end to visualisation software that are supported inside the platform. Any visualisation software running either on external systems or on client browsers do not need to interact with this microservice. They can directly use the data provided by the data microservice.

DTaaS is a collaborative platform. Users are able to have private and shared storage space to enable controlled sharing of either DT assets or DTs. The re-usable assets microservice (Asset MS) provides search, explore, and select functions over DT assets. Thus Asset MS should aid users in performing create-read-update-delete operations on the private and shared re-usable assets. Any ready to use DTs are also made available via the Asset MS.

The DT life cycle microservice assists users during all life phases of a DT. This microservice extensively uses other microservices to provide atomic operations at the level of DTs. This microservice acts as a controller to both the Asset and Exec microservices.

The execution manager microservice (Exec MS) is responsible for on-demand provisioning of virtual compute infrastructure. To make the platform scalable, the Exec MS must be capable of integrating with private and public cloud providers. Users operate with these isolated workspaces. The Exec MS receives DT configuration from DT life cycle MS and executes DT in isolated compute environment (ex: containers, virtual machines etc.). The Exec MS must interpret DT configuration specified in Definition 2. The required compute environment shall be provisioned as specified in C_i. The C_e and C_{pt} are connections to external software and PT respectively. These connections are to be managed as network access control policies. Finally, C_a helps with gathering of required DT assets. The data assets managed by Asset MS are linked to the DT using access credentials provided in C_a. The Exec MS also downloads the required model, function and tool assets and then places them in the compute environment. If all the above steps are successful, DT becomes available for use. The DT life cycle MS takes over the management of all the DTs provisioned by Exec MS and takes them through life cycle phases as per the events triggered on the respective DTs. The Exec MS comes back into reckoning during the termination life cycle phase of a DT.

Platform Implementation

The current implementation of DTaaS supports re-usable assets, DT life cycle, user workspaces and providing DTs as a service within the platform. Figure 9 shows the functioning system components providing the features mentioned here. There have been five releases of software so far and it is actively used by researchers and engineers from software engineering, electrical, civil, mechanical and robotics domains. The entire DTaaS platform is available as two containerised packages. One is for the core platform and another is for the platform services. Users only need configure the OAuth, asset location and network host/port information and the application becomes installable. There is also an DTaaS CLI python package[78] to perform user management of the platform.

The current security functionality is based on mutual Transport Layer Security (mTLS) and OAuth2 protocols. Users receive signed TLS certificates. The TLS certificate-based mutual TLS (mTLS) authentication and OAuth2 provides better security than the usual username and password combination. The mTLS authentication takes place between the users browser and the platform gateway. The
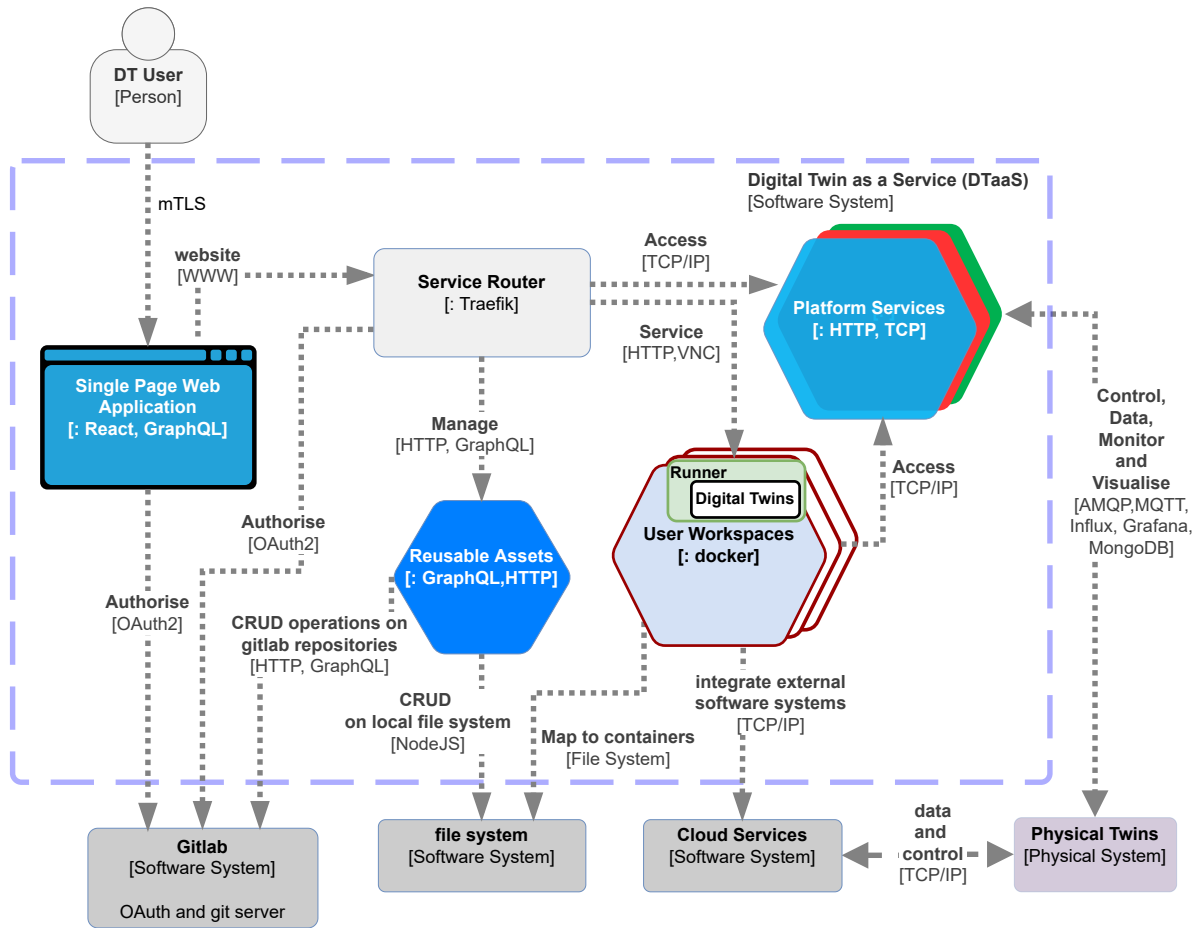
Figure 9. The current implementation of DTaaS platform[76]. It supports composable DTs and provides integrated user workspaces. Re-usable assets microservice performs create, read, update and delete (CRUD) operations on DT assets.
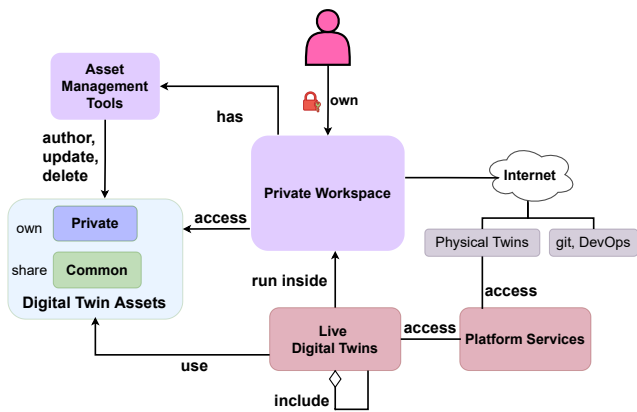

Figure 10. A relational view of the user workspace[79]. The system components are all linked to the workspace and the DTs run inside the workspace. The DTs are also offered as services to internal and external users.

gateway federates all the back-end services. The service discovery, load balancing, and health checks are carried by the service based on a dynamic reconfiguration mechanism. The OAuth2 integration is active for both the website and other services accessed via the service router.

Each DTaaS software installation comes with support for multiple platform services. These services are available to all users, DTs running inside the platform and the PTs linked to the DTs running within the platform. The DTs

and PTs access the platform services using the standard network (TCP/IP) protocols. Thus any services supporting these protocols can be integrated with DTaaS. Five services have been integrated with the platform so far primarily to support the case studies already implemented on the platform. The integrated services provide communication, data storage, monitor and visualisation functionalities.

InfluxDB[80] time-series database and MongoDB[81] document database are also integrated into the platform. InfluxDB comes with data storage, and querying and visualisation dashboards. The incubator case study uses this platform service for data storage and visualisation. Users are permitted to share the dashboards. Thus DT experts can develop custom dashboards and share them with other users. MongoDB[81] Java Script Object Notation (JSON) document database is now part of DTaaS. If DTs or users wish to operate using data or asset configuration expressed in JSON, there is support for such usage.

There is support for AMQP[82], MQTT[83] and OPC-UA[84] protocol-based communication between PT and DT. These communication-related platform services can be used for bidirectional transfer of data and control commands between DT-PT pairs. Thirteen exemplar case studies have been made available publicly to showcase this support[85] out of which seven case studies have been summarised in this paper.

The Grafana[86] has been integrated as another platform service. This service is a de facto industry software to monitor, visualise, observe and trace distributed systems.

Given the need to support PTs deployed across multiple physical locations, software like Grafana is a good choice for monitoring and visualisation. These services can be provided with data from PT in which case there is only PT monitoring and visualisation. Digital Shadows provide one way communication from PT to DT. Even in this case, the Grafana and other platform services can be used to provide monitoring and visualisation services. This same observation is valid even for DTs. A development version of the firefighter case study uses Grafana dashboard for monitoring and visualisation purposes.

A *re-usable assets* microservice has been developed to provide access to re-usable DT assets to the platform users. This microservice uses GraphQL protocol for asset discovery and HTTP protocol for file transfer. The users can store both private DT assets and also get access to shared DT assets. Users can synchronise their private DT assets with external Git repositories. In addition, asset repository transparently gets mapped to user workspaces within which users can perform DT life cycle operations.

Many DTs do not provide API interfaces. Such a limitation reduces the integration possibilities of DTs with other DTs, platform services and external software systems. In order to improve the integration possibilities, a *runner* microservice has been developed to create REST API wrappers around DTs. This microservices converts REST API calls to invoke life cycle scripts or any other scripts that can support direct operations on DTs. A user can create many DTs with or without runner integration. DTs with runner integration will be available as service to other users of DTaaS. Thus users have the flexibility to hide proprietary/private DTs and only provide access to them as services to other users. This feature gives rise to the possibility of creating collaborative DTs that are managed by multiple users.

A YAML Ain't Markup Language (YAML)[87] based configuration format is been defined to specify the DT configuration explained in Digital Twin Definition section. This configuration format will be used in future by the DT Exec MS. The users can fill the configuration template from their workspace or via web services exposed via the single page application.

All users have dedicated workspaces. The workspaces are implemented as docker containers running desktop Linux environment. These come with isolated docker containers having streaming desktop, VSCode, Jupyter notebooks and remote terminal access.

A relational view of user workspace is provided in Figure 10. Each user workspace is private and is securely available via unified interface integrated into the single page web application. Users have access to re-usable DT assets. Users can install asset management tools of their own in the private workspace and use them to manage their DT assets. Users can also update the configuration of a DT and run life cycle scripts.

Users can run DTs within their workspace and make them accessible to other users. Any DT capable of providing services can be accessed by other users as services, thereby making DTs available as services.

Users can also permit remote access to live DTs via platform services. If a DT does not have an API interface, runner can be used to provide one which makes the DT becomes available as a service within and outside of user workspace. Using the service API interfaces of DTs with or without runner, users can treat live DTs as service components in their own software systems.

In addition, these workspaces have Internet access. Thus, the PT to DT bidirectional communication link is as simple as spawning required client-server communication protocol software. In addition, the platform services can always be used for bidirectional PT-DT communication. It is possible to restrict the DT-PT and DT-Internet communication. A suitable network firewall configuration can easily enforce the necessary restrictions. If users are well-versed in the software management processes, DevOps related techniques can be used from within the workspaces.

Model Management

It is important for DTs and DT platforms to support a combination of models produced using different mathematical paradigms[88]. DTs are heterogeneous of nature and thus when DTs are established for CPSs this support is paramount. In order to accommodate such integration we make use of the functional mockup interface[89] (FMI) standard but naturally there are other alternatives as well. The DTaaS already supports such diverse modelling paradigms. This support is made possible in four ways.

First is the availability of user workspaces in which models can be created and managed. Users are at liberty to use programming languages, engineering tools and frameworks of their choice for model creation and management. The model generation tools such as Modelica, Gmsh[90] and Matlab can be installed and used for creation of models. Such tools can also help with the simulation of DTs in which these models are included. Thus models of different formats and representations are already supported inside the DTaaS. The examples presented in Table 3 already have finite element method (FEM), physics, ordinary differential equation-based models published in FMUs, Modelica, and Python package representations. Other commercial partners have demonstrated proprietary System of Systems and FEM models as Matlab and Geo (by the Gmsh tool) files. These proprietary models could not be described in this paper due to commercial considerations.

Second is the just-in-time (JIT) compilation of models. Sometimes users create models in one format (say in Modelica format) and compile them to another format (say FMUs) before the model is used inside a DT. Model compilation scripts used in the create DT life cycle phase (explained in Phases in Digital Twin Life Cycle) a DT perform this kind of JIT compilation. The firefighter case study takes advantage of this technique to compile a Modelica representation into an FMU.

Third is the ability to use complementary and replaceable models for a single DT. The Re-usable Digital Twin Assets details different models in the incubator and the firefighter case studies. The selection of the available models for a DT is specified in the DT configuration.

Fourth is the dynamic, run-time integration of models into DTs. In this case, model integration and/or swapping happens during the evolve phase of a DT. This run-time change of DT models is controlled by DT configuration and

data received by the DT at run-time. The Water tank model swap example demonstrates dynamic swapping of FMUs.

While DTaaS allows the technical combination of different kind of models, it is in the responsibility of the DT creator to assure a semantically meaningful combination. Simulation granularity, especially notion of time, but also to the physical units, etc. has to be ensured when building a DT instance.

Comparison with Existing Platforms

Prior work[17,44] provides qualitative comparison of DT platforms based on *Structured data*, *Language specification*, *Co-simulation*, *Domain-specific*, and *Geospatial data* DT frameworks and platforms. In order to objectively assess DTaaS proposed in this work, we take similar criteria to compare DTaaS with some existing frameworks and platforms that are used to realise DTs in the categories of *Structured data*, *Language specification*, and *Co-simulation*. The other two categories, namely, *Domain-specific* and *Geospatial data*, are not considered since they are out of the scope of the DTaaS platform.

Since some of the frameworks presented in previous surveys[17,44] share similar characteristics, e.g., AAS-based implementations (Eclipse BaSyX, SAP I4.0 AAS, NOVAAS, PYI40AAS, and AASX Package Explorer), IoT-based implementations (Eclipse Ditto, Azure DTs, and AWS IoT Greengrass), languages (DTDL, Eclipse Vorto, and Twined), and model-based implementations (INTO-CPS Co-simulation Framework and CPS-Twinning), a representative sample is taken with the most complete framework(s) per group based on the authors' knowledge and experience with the tools and frameworks. Thus, the comparison considers Eclipse BaSyX and Eclipse Ditto in the *Structured DT* category, DTDL and Eclipse Vorto in the *Language specification* category, and INTO-CPS Co-simulation framework in the *Co-simulation* category. We compare the capabilities of DTaaS with the selected frameworks in a range of features, such as its capabilities regarding configuration files, which is relevant in frameworks like DTDL[36], Eclipse Vorto[91], and the AAS schema; its platform capabilities, which is relevant in frameworks like Eclipse BaSyX[41] and Eclipse Ditto[39]; its modelling and simulation capabilities, which is relevant in frameworks like the ones in the *Structured DT* and *Co-simulation* categories; and finally, its capabilities to integrate services both infrastructure- and DT optimisation-wise in comparison with the three categories mentioned above. Table 2 provides an overview of the comparison for the criteria *Description of DTs and structures*, *Composition*, *DT-to-DT (DT-DT) relationships*, *Default bi-directional synchronisation*, *Running simulators*, *Binding to infrastructure services*, *Binding to DT (optimisation) services*, and *Re-usability*, inspired by the existing surveys on DT platforms/frameworks[17,44,92].

In comparison to the selected platforms, as detailed in Table 2, DTaaS approach performs well in terms of the bi-directional communication capabilities, DT description, re-usability, binding to infrastructure services, and more importantly, integrating running simulators. Yet, it struggles with incorporating DT-to-DT relationships, especially for a more semantically accurate way to compose DTs, and with defining a more straightforward mechanism to set up and modularly attach DT (optimisation-wise) services. Additionally, DTaaS supports hosting external third-party

DT frameworks, such as the *DT Manager*[47], where such an approach becomes a *Tool* in DTaaS context. This way, DTaaS can be used as a hub converging the realisation of DTs where multiple tools, frameworks, and design paradigms co-exist.
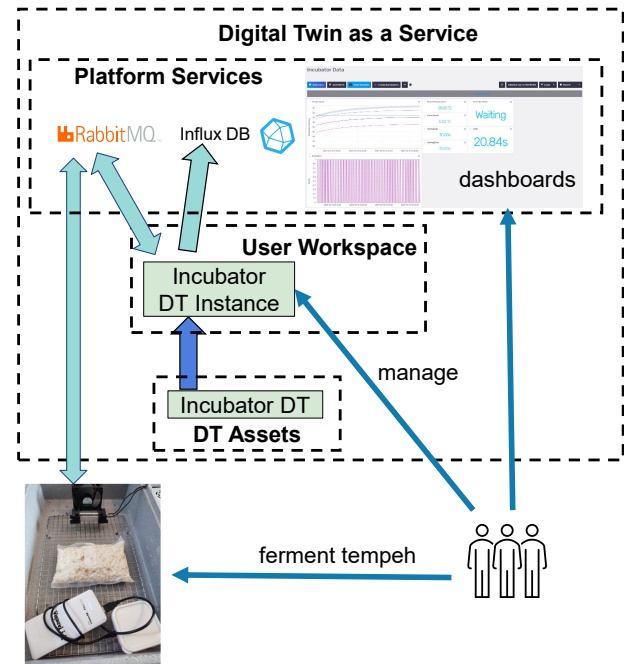

Figure 11. Incubator DT in DTaaS platform.

Implementation of Case Studies

This section demonstrates use of DTaaS for working with DTs.

Incubator

The incubator was originally developed independently from DTaaS platform. This DT is an example of developing DTs outside of DTaaS and deploying them on it. The DT itself is re-usable in other DTs but does not have re-usable DT assets included in it.

Figure 11 shows the implementation of the incubator DT inside DTaaS. User instantiates an already available DT in the workspace as per the configuration. The DT connects with PT via RabbitMQ service and publishes time-series data to InfluxDB. A dashboard has been setup on InfluxDB to showcase analytical results produced by the DT. Please see prior work[94] for more explanation of the analytical results.

Firefighter

The firefighter DT case study implementation takes advantage of the DT concepts supported in DTaaS. The firefighter DT is put together from data, model, function and tool assets as described in the Firefighter sub-section. The DT uses MQTT, a lightweight messaging protocol, to handle data transmission. This broker acts as the central point for data exchange, ensuring soft real-time and reliable communication among various system elements without the need for synchronisation. The DT also uses data storage (InfluxDB) and visualisation (Grafana) services integrated into the platform. Grafana provides a Graphical

Table 2. Summary of comparison of existing platforms. The qualitative criterion has been adopted from prior work[17,44,47].

Criterion	Structured DT		Language specification		Co-simulation	This approach
	Eclipse BaSyX (AAS)	Eclipse Ditto	DTDL	Eclipse Vorto	INTO-CPS Co-simulation framework	
Description of DTs and structures	By AAS *Sub-models*, mainly defined by *Properties* and *Operations*	By JSON structures, defined by *Attributes*, *Features*, and *Definitions*	By DTDL-based JSON structures, defined by *Commands*, *Components*, *Properties*, *Relationships*, and *Telemetries*	By Vorto *Information Models*, composed of object-oriented *Function Blocks*, including *Properties*, *Events*, and *Operations*	By the Maestro Multi-model configuration file, mainly defined by *FMUs*, *Connections*, and *Parameters*	By the DT configuration, defined by *Assets* — which can contain *Data*, *Models*, *Functions*, *Tools* and **Digital Twins** — *Infrastructure*, *Physical Twins*, and *Services*
Composition	By aggregation of *Sub-models*	By aggregation of *Features*	By composing *Interfaces* via *Components* - and composition via *Relationships*-	By aggregation of *Function Blocks*	By aggregation of hierarchical FMU simulators	By aggregation of *Assets*
DT-DT relationships	Implicitly by semantic identifiers via *References*	N.A.	Explicitly by the *Relationships* field	Implicitly by *Model References*	N.A.	Only applicable for DT composition
Default bi-directional synchronisation	As a DT	as a DT	As a Digital Model (requires a back-end to set up a DT via *Telemetries* and *Commands*)	As a Digital Model (requires a back-end to set up a DT via *Properties*, *Operations*, and *Events*)	As a Digital Model (requires RabbitMQ FMU[93] to set up a DT)	As a DT
Running simulators	No	No	No	No	Yes (via FMUs and Maestro)	Yes (multiple simulators included as reusable tools)
Binding to infrastructure services	Provides APIs to support OPC UA, MongoDB, PostgreSQL, Docker, HTTP REST, AutomationML, and IEC 61360 product dictionaries (e.g. ECLASS)	Provides APIs to support Apache Kakfa, MongoDB, nginx, HTTP REST, AMQP, MQTT, and WebSocket	Compatible with Microsoft Azure DTs	Compatible with Eclipse Ditto and Eclipse Hono	Dependent on services wrapped as FMUs or accessible via the FMI interface	Provides a set of infrastructure services that are initialised from the *Infrastructure* field in the DT configuration
Binding to DT (optimisation) services	Exposes APIs that can be used by external services	Exposes APIs that can be used by external services	N.A.	N.A.	Dependent on pre-defined FMUs that include such services (not necessarily generalisable to multiple case studies)	Can be done through (1) internal or external infrastructure services and (2) a combination of *Functions* and *Tools* and the business logic in the corresponding life cycle phase
Reusability	Of *Sub-models* and infrastructure	Of *Features* and infrastructure	Of *Components* and *Interfaces*	Of *Function Blocks*	Of FMUs	Of *Data*, *Models*, *Functions*, *Tools*, and infrastructure

User Interface (GUI) for decision making by the mission commander. A TeSSLa monitor is implemented for real-time monitoring of both the time needed to leave the building and breathing time left in the SCBA. It alerts when predefined thresholds are reached. The Telegraf-TeSSLa-connector[95] integrates TeSSLa into the system.
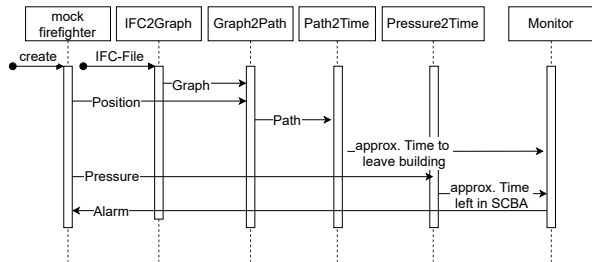

Figure 12. Sequence diagram of the DT for the rescue mission scenario

Figure 12 provides an overview of the sequence of events, facilitating efficient data exchange and integration of different components. There are two sequences, each of which calculates a time. A graph is generated once in `IFC2Graph` with the 3D building file, in which navigation then takes place. The `Graph2Path` uses the position of the mock firefighter to calculate the path to the next exit. This path is used in `Path2Time` to approximate the time required to leave the building along a given path. The reported pressure values of the mock firefighters SCBA is used in `Pressure2Time` to approximate the time for which oxygen is available. These two times are monitored continuously. If there is a risk of insufficient oxygen available to leave the building, the firefighter is warned.

The firefighter DT has been validated through simulations and feedback from firefighter technology providers involved in the O5G-N-IoT project as well as a few firefighters. These stakeholders have provided valuable insights into practical requirements and potential improvements for the system.

The firefighter DT is run in user workspace. DTaaS supports multiple firefighters DTs for each user while still maintaining the required isolation between all the active firefighter DTs. A collective monitoring results for different firefighter DTs can also be shown as one single visualisation dashboard.

Other Exemplar Case Studies

The existing users of DTaaS have developed multiple Digital Shadows and DTs to support their case studies. They have also generously contributed non-proprietary versions of their case studies as exemplars to other users of DTaaS[85]. Seven of these exemplars are summarised in Table 3. The exemplars comes from diverse applications domains including Physics, Robotics, food industry and water systems. The Desktop Robotti is a DT created for an autonomous agricultural vehicle[96]; the Flex-Cell Robots is a DT for flex-cell workstation[97]. The mass spring damper with runtime monitor demonstrates the use of NuRV[71] as a runtime verification (RV) and monitoring tool. This RV tool is integrated into the mass spring damper co-simulation digital model (DM)[98]. There is no PT for this DM. However, this exemplar demonstrates use of two tools, namely NuRV

and Maestro[99] for creating a DM. Such use of multiple models and tools ín one DT has been demonstrated again in other exemplars.

The Water tank fault injection[100] and model swap[101] demonstrate the advantages of run-time fault injections and dynamic model swapping in DT.
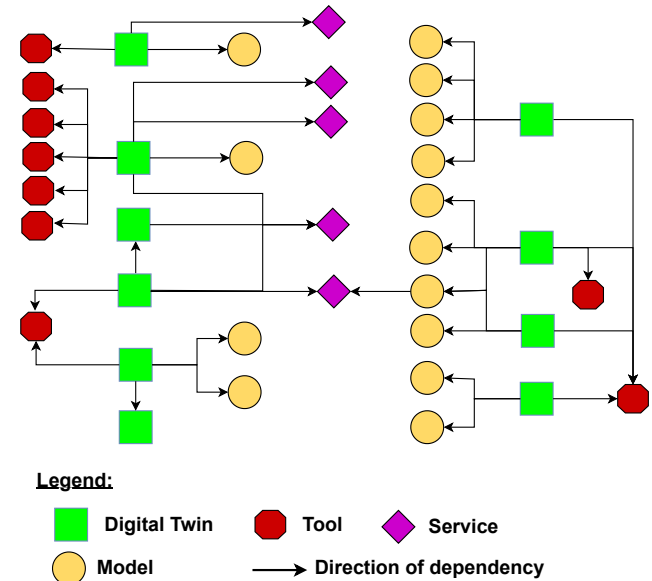

Figure 13. A dependency graph of all re-usable assets included in the case studies quoted in this paper. The dependency arrow goes from composing asset to composed asset.

There is a significant reuse of DT assets among the DTs created in different application domains. For example, the DT Robotti, Flex-Cell Robots and Water tank model swap case studies use Maestro co-simulation engine as a DT tool asset. There is even reuse of models among the water tank model swap and water tank fault injection case studies. The incubator with NuRV run-time monitor service is a DT of incubator with integrated run-time monitoring service. This DT is an example of reusing one DT inside another DT. The Flex-Cell Robots is a pair of robots each of which have a unique DT which in turn are included in the Flex-Cell Robots DT. Thus it is possible to use more than one DT inside one DT and DTaaS supports such use cases as well.

Quite a few of these exemplars use the platform services integrated into DTaaS. The Desktop Robotti, Flex-Cell Robots and incubator with NuRV monitor service use RabbitMQ service. The Flex-Cell Robots also uses MQTT service. The waste water treatment exemplar uses OPC-UA communication service. Thus the advantages of integrated platform services is evident in supporting the diverse case studies within DTaaS.

The Figure 13 illustrates the potential for re-usability of assets in DTs constructed using composition technique. The composition can be applied at either model/tool/service-level or at the DT level. There are two instances of a DT being composed in another DT. Such reuse is completely supported in DTaaS. Another advantage of this approach is to tailor the platform usability to the expertise of users. The expert users can create and manage all re-usable assets including DTs. Each DT is specified using one DT configuration stored in YAML format. The users of DT can create multiple instances

Table 3. The implementation of seven more DT exemplar case studies inside DTaaS.

Exemplar Name	Application Domain	Physical Twin - Real / Mock / Both	Reuse of:	life cycle phases
Desktop Robotti	Robotics	Both	Model, Tool and Service	create, execute, terminate
Flex-Cell Robots	Robotics	Both	Models, Tool and Services	create, prepare (configures all assets from one DT configuration), execute, save, analyse, terminate
Incubator with NuRV run-time monitor service	Food Safety	Both	Models and Services	create, execute, analyse, terminate
Mass Spring Damper with runtime monitor	Physics	None	Models and Tool	create, execute, analyse, terminate
Water tank fault injection	Water	Mock	Models and Tools	create, execute, analyse, terminate
Water tank model swap	Water	Mock	Models and Tool	create, execute, evolve (auto), analyse, save, terminate
Waste water treatment	Water	Both	Functions, Services, Tool	create, execute, terminate

of DT by having multiple configurations one for each DT instance. Thus users of DT need not concern themselves with the internal details of DT; they can just modify the DT configuration and instantiate a live DT which can then support the services built on top of it. These services can either be platform services or external services. The non-expert and decision makers can utilise the analytical results produced by the services.They do not need to even manage the DT itself; access to DT-supported services is sufficient.

Performance Evaluation

Incubator

Multiple incubator DTs can be run by one user. Multiple users can run many incubators concurrently. The only limitation is on the capabilities of the underlying hardware to support the number of live incubators DTs. The incubator has a sensor that samples temperature values every 3 seconds. Thus the network delays and packet losses below are not observable.

Firefighter

In our performance evaluation, we conducted two experiments related to the firefighter use case to assess the system's efficiency and robustness.

The first experiment involved comparing the performance of platform-hosted and remote MQTT servers. To compare, the system was configured to use either the integrated MQTT server or remote MQTT server and the communication times to and from a mock PT were measured with an additional MQTT subscriber. For each experiment 500 measurements were taken. The results are summarised in Table 4. The findings indicate that the integrated MQTT service significantly improves the system's performance compared to the remote MQTT server.

Table 4. Communication times for local and remote MQTT servers with and without network stress averaged over 500 measures each

Performance evaluation		
Configuration	Avg. time	Std. deviation
Local MQTT (no iPerf)	1047 ms	278 ms
Remote MQTT (no iPerf)	1732 ms	465 ms
Local MQTT (with iPerf)	1795 ms	269 ms

The second experiment focused on the impact of network stress on DTs running inside DTaaS. The network stack on the server hosting DTaaS has been kept busy by iperf[102] throughput testing tool. The TCP client and server of the throughput test are on the same server. Thus the effective network bandwidth available to the DTs becomes reduced. The results are shown in Table 4. Despite the network stress, both the DT and DTaaS continued to perform as per the requirements, demonstrating its robustness under adverse conditions. The firefighter DT does not have safety and reliability system in place yet. We are currently working on a Fault Detection Identification and Recovery (FDIR) solution for some properties of the system. We intend to monitor the 5G configuration, possibly with some mesh network between the User Equipment (mobile devices) and the data rate usage of different data streams to detect faults, try to identify their causes and recover from them.

Discussion and Future Work

DTaaS is a platform for building, using, and sharing DTs. The reuse of DT assets is a guiding principle around which DTs are managed within DTaaS. This approach has been illustrated with two detailed case studies, and seven additional exemplar case studies. Significant progress has been made in the platform implementation, specifically

in developing the reusable assets microservice, enhancing scalability. The existing data, models, and tools have been shown to be reusable, facilitating a smooth transition for users from their local development environments to DTaaS hosted on the cloud.

Additionally, the DT configuration format has been defined and mapped to the two primary case studies, aiding new users in creating their own DTs and simplifying the adoption of DTaaS. A performance evaluation on the firefighter case study showcases DTaaS's capability to handle PTs operating both on-premise and remotely, discusses the impact of network latency and validates the platform's ability to execute multiple DTs concurrently. Five versions of DTaaS software have been released and it is being actively used by researchers and commercial partners.

Ongoing work includes integrating semantic modules as a DTaaS platform service, enabling the interpretation of DTs as a knowledge graph for consistency checking and semantic querying. Future work will focus on extending the platform's capabilities, enhancing the semantic module, and further validating DTaaS approach through additional case studies and user feedback.

Concluding Remarks

There is a strong interest in the DT community to provide Digital Twin as a Service platform (DTaaS) to spread the user base of DTs. A typical DT life cycle on DTaaS involves create, execute, save, analyse, evolve and terminate phases. Only software platforms developed with awareness of DT life cycle can aspire to fulfil DTaaS vision. Re-usability of DT assets, creation of meaningful DT configuration, scalable deployment are key challenges in the development of DTaaS platforms.

In this paper, we describe some nuances in DT configuration that is valid in the context of re-usable assets, shared infrastructure, and desired integration with external world. We believe that the ability to reuse DT assets on DTaaS will make it cheaper and easier to get started with DTs.

Supplemental material

The source code for the Digital Twin as a Service software is available at: *https://github.com/INTO-CPS-Association/DTaaS*.

Declaration of conflicting interests

The authors declare no conflict of interest.

References

1. Böttjer T, Tola D, Kakavandi F et al. A review of unit level digital twin applications in the manufacturing industry. *CIRP Journal of Manufacturing Science and Technology* 2023; 45: 162–189. DOI:10.1016/j.cirpj.2023.06.011.

2. Naseri F, Gil S, Barbu C et al. Digital twin of electric vehicle battery systems: Comprehensive review of the use cases, requirements, and platforms. *Renewable and Sustainable Energy Reviews* 2023; 179: 113280. DOI:10.1016/j.rser.2023.113280.

3. Tao F, Xiao B, Qi Q et al. Digital Twin Modeling. *Journal of Manufacturing Systems* 2022; 64: 372–389. DOI:10.1016/j.jmsy.2022.06.015.

4. Zambrano V, Mueller-Roemer J, Sandberg M et al. Industrial digitalization in the industry 4.0 era: Classification, reuse and authoring of digital models on digital twin platforms. *Array* 2022; : 100176.

5. Lehner D, Pfeiffer J, Tinsel E et al. Digital twin platforms: Requirements, capabilities, and future prospects. *IEEE Software* 2022; 39(02): 53–61. DOI:10.1109/MS.2021.3133795.

6. Abburu S, Berre AJ, Jacoby M et al. Cognitwin–hybrid and cognitive digital twins for the process industry. In *2020 IEEE International Conference on Engineering, Technology and Innovation (ICE/ITMC)*. IEEE, pp. 1–8.

7. Aziz A, Chouhan SS, Schelén O et al. Distributed digital twins as proxies-unlocking composability & flexibility for purpose-oriented digital twins. *IEEE Access* 2023; .

8. Aheleroff S, Xu X, Zhong RY et al. Digital Twin as a Service (DTaaS) in Industry 4.0: An Architecture Reference Model. *Advanced Engineering Informatics* 2021; 47: 101225.

9. Feng H, Gomes C, Gil S et al. Integration Of The Mape-K Loop In Digital Twins. In *2022 Annual Modeling and Simulation Conference (ANNSIM)*. IEEE. DOI:10.23919/annsim55834.2022.9859489.

10. [O5G-N-IoT] Campusnetzwerke für Krisenszenarien, 2024. URL https://o5g-n-iot.de/. Last accessed 06 June 2024.

11. Talasila P, Gomes C, Mikkelsen PH et al. Digital twin as a service (dtaas): A platform for digital twin developers and users. In *Digital Twin 2023: The 2023 IEEE International Conference on Digital Twin*. Portsmouth, UK: IEEE.

12. Grieves M and Vickers J. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In Kahlen FJ, Flumerfelt S and Alves A (eds.) *Transdisciplinary Perspectives on Complex Systems*. Springer International Publishing Switzerland, pp. 85–113.

13. Feng H, Gomes C, Gil S et al. Integration of the mape-k loop in digital twins. doi: 10.23919/ANNSIM55834.2022.9859489: IEEE.

14. Oakes BJ, Gomes C, Larsen PG et al. Examining model qualities and their impact on digital twins. In *2023 Annual Modeling and Simulation Conference (ANNSIM)*. IEEE, pp. 220–232.

15. Kritzinger W, Karner M, Traar G et al. Digital Twin in Manufacturing: A Categorical Literature Review and Classification. In *IFAC-PapersOnLine*, volume 51. Elsevier B.V., pp. 1016–1022. DOI:10.1016/j.ifacol.2018.08.474.

16. VanDerHorn E and Mahadevan S. Digital Twin: Generalization, characterization and implementation. *Decision Support Systems* 2021; 145. DOI:10.1016/J.DSS.2021.113524.

17. Gil S, Mikkelsen PH, Gomes C et al. Survey on open-source digital twin frameworks–A case study approach. *Software: Practice and Experience* 2024; 54(6): 929–960. DOI:10.1002/spe.3305.

18. Oakes B, Parsai A, Mierlo SV et al. Improving digital twin experience reports. In *Proceedings of the 9th International Conference on Model-Driven Engineering and Software Development*. SCITEPRESS - Science and Technology Publications. DOI:10.5220/0010236101790190.

19. Lee J, Bagheri B and Kao HA. A Cyber-Physical Systems Architecture for Industry 4.0-based Manufacturing Systems. *Manufacturing Letters* 2015; 3: 18–23. DOI:10.1016/J.MFGLET.2014.12.001.

20. Lu Y, Liu C, Kevin I et al. Digital twin-driven smart manufacturing: Connotation, reference model, applications and research issues. *Robotics and computer-integrated manufacturing* 2020; 61: 101837.

21. Dalibor M, Heithoff M, Michael J et al. Generating Customized Low-Code Development Platforms for Digital Twins. *Journal of Computer Languages* 2022; 70: 101117. DOI:10.1016/J.COLA.2022.101117.

22. Talasila P, Crăciunean DC, Bogdan-Constantin P et al. Comparison Between the HUBCAP and DIGITBrain Platforms for Model-Based Design and Evaluation of Digital Twins. In *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, volume 13230 LNCS. ISBN 9783031124280, pp. 238–244. DOI:10.1007/978-3-031-12429-7_17.

23. DIGITbrain establishing a modular Approach to Digital Twins for Manufacturing, 2024. URL https://digitbrain.eu/. Last accessed 06 June 2024.

24. Mao R, Li Y and Zhang H. Digital twin-based research in the maritime industry: A brief survey. In *IECON 2023-49th Annual Conference of the IEEE Industrial Electronics Society*. IEEE, pp. 1–6.

25. Liu C, Zheng P and Xu X. Digitalisation and servitisation of machine tools in the era of industry 4.0: a review. *International journal of production research* 2023; 61(12): 4069–4101.

26. Zhang H, Li G, Hatledal LI et al. A digital twin of the research vessel gunnerus for lifecycle services: Outlining key technologies. *IEEE Robotics & Automation Magazine* 2022; .

27. Petrova-Antonova D, Spasov I, Krasteva I et al. A digital twin platform for diagnostics and rehabilitation of multiple sclerosis. In *Computational Science and Its Applications–ICCSA 2020: 20th International Conference, Cagliari, Italy,* *July 1–4, 2020, Proceedings, Part I 20*. Springer, pp. 503–518.

28. McKee D. Platform stack architectural framework: An introductory guide. *A Digital Twin Consortium White Paper Digital Twin Consortium* 2023; .

29. Hogan A, Blomqvist E, Cochez M et al. Knowledge graphs. *ACM Comput Surv* 2022; 54(4): 71:1–71:37.

30. Sahlab N, Kamm S, Müller T et al. Knowledge graphs as enhancers of intelligent digital twins. In *ICPS*. IEEE, pp. 19–24.

31. Waszak M, Lam AN, Hoffmann V et al. Let the asset decide: Digital twins with knowledge graphs. In *ICSA Companion*. IEEE, pp. 35–39.

32. Zheng X, Lu J and Kiritsis D. The emergence of cognitive digital twin: vision, challenges and opportunities. *Int J Prod Res* 2022; 60(24): 7610–7632.

33. Li Y, Chen J, Hu Z et al. Co-simulation of complex engineered systems enabled by a cognitive twin architecture. *Int J Prod Res* 2022; 60(24): 7588–7609.

34. Li H, Wang G, Lu J et al. Cognitive twin construction for system of systems operation based on semantic integration and high-level architecture. *Integr Comput Aided Eng* 2022; 29(3): 277–295.

35. Pfeiffer J, Lehner D, Wortmann A et al. Modeling Capabilities of Digital Twin Platforms-Old Wine in New Bottles? *Journal of Object Technology* 2022; 21(3). DOI:10.5381/jot.2022.21.3.a10.

36. Digital Twins Definition Language, 2024. URL https://github.com/Azure/opendigitaltwins-dtdl. Last accessed 06 June 2024.

37. Vorto Language for Digital Twins, 2024. URL https://github.com/eclipse/vorto/blob/development/docs/vortolang-1.0.md. Last accessed 06 June 2024.

38. Connect, Command & Control IoT devices :: Eclipse Hono, 2024. URL https://eclipse.dev/hono. Last accessed 06 June 2024.

39. Eclipse Ditto: open source framework for digital twins in the IoT, 2024. URL https://eclipse.dev/ditto. Last accessed 06 June 2024.

40. Digital Twins Made Easy — AWS IoT TwinMaker — Amazon Web Services, 2024. URL https://aws.amazon.com/de/iot-twinmaker/. Last accessed 06 June 2024.

41. Eclipse BaSyx, 2024. URL https://eclipse.dev/basyx/. Last accessed 06 June 2024.

42. Plattform Industrie 40. Reference Architectural Model Industrie 4.0 (RAMI 4.0) - An Introduction. Technical report, ZVEI - German Electrical and Electronic Manufacturers Association, 2016. URL https://web.archive.org/web/20210615073607/https://www.plattform-i40.de/PI40/Redaktion/EN/Downloads/Publikation/rami40-an-introduction.pdf?{_}{_}blob=publicationFile{&}v=7.

43. IEC. *Asset Administration Shell for industrial applications - Part 1: Asset Administration Shell structure*. IEC 63278-1:2023 ed. Geneva, Switzerland: International Electrotechnical Commission, 2023. URL https://webstore.iec.ch/publication/65628.

44. Lehner D, Pfeiffer J, Tinsel EF et al. Digital Twin Platforms: Requirements, Capabilities, and Future Prospects.

IEEE Software 2022; 39(2): 53–61. DOI:10.1109/MS.2021.3133795.

45. Ferko E, Bucaioni A, Pelliccione P et al. Standardisation in digital twin architectures in manufacturing. In *2023 IEEE 20th International Conference on Software Architecture (ICSA)*. IEEE, pp. 70–81.

46. Ferko E, Bucaioni A and Behnam M. Architecting digital twins. *IEEE Access* 2022; 10: 50335–50350.

47. Lehner D, Gil S, Mikkelsen PH et al. An architectural extension for digital twin platforms to leverage behavioral models. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. pp. 1–8. DOI: 10.1109/CASE56687.2023.10260417.

48. Stojanovic L, Usländer T, Volz F et al. Methodology and tools for digital twin management—the fa3st approach. *IoT* 2021; 2(4): 717–740.

49. Steindl G and Kastner W. Semantic Microservice Framework for Digital Twins. *Applied Sciences* 2021; 11(12). DOI: 10.3390/app11125633.

50. Ciavotta M, Maso GD, Rovere D et al. Towards the Digital Factory: A Microservices-based Middleware for Real-to-Digital Synchronization. In *Microservices: Science and Engineering*. Springer International Publishing. ISBN 9783030316464, 2019. pp. 273–297. DOI:10.1007/978-3-030-31646-4_11.

51. Reiterer SH, Balci S, Fu D et al. Continuous Integration for Vehicle Simulations. In *2020 25th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, volume 1. IEEE, pp. 1023–1026.

52. Beetz F and Harrer S. GitOps: The Evolution of DevOps? *IEEE Software* 2021; 39(4): 70–75.

53. Feng H, Gomes C, Thule C et al. Introduction to Digital Twin Engineering. In *2021 Annual Modeling and Simulation Conference (ANNSIM)*. IEEE. DOI:10.23919/annsim52504.2021.9552135.

54. Convent L, Hungerecker S, Leucker M et al. Tessla: Temporal stream-based specification language. In Massoni T and Mousavi MR (eds.) *Formal Methods: Foundations and Applications*. Cham: Springer International Publishing. ISBN 978-3-030-03044-5, pp. 144–162.

55. TeSSLa: A Convenient Language for Specification and Verification of Your System, 2024. URL https://www.tessla.io/. Last accessed 06 June 2024.

56. Steinkraus KH, Hwa YB, Van Buren JP et al. Studies on tempeh. An Indonesian fermented soybean food. *Food Research* 1960; 25: 777–788.

57. Leucker M, Sachenbacher M and Vosteen LB. Digital twin for rescue missions–a case study. In *Proceedings of the FMDT 2023 Workshop on Applications of Formal Methods and Digital Twins*. CEUR Vol-3507.

58. Esterle L, Gomes C, Frasheri M et al. Digital Twins for Collaboration and Self-integration. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*. IEEE. DOI: 10.1109/acsos-c52956.2021.00040.

59. Functional Mock-up Interface, 2024. URL https://fmi-standard.org/. Last accessed 06 June 2024.

60. Thule C, Lausdahl K, Gomes C et al. Maestro: The INTO-CPS Co-simulation Framework. *Simulation Modelling Practice and Theory* 2019; 92: 45 – 61. DOI:https://doi.org/10.1016/j.simpat.2018.12.005. URL http://www.sciencedirect.com/science/article/pii/S1569190X1830193X.

61. OpenFOAM, 2024. URL https://www.openfoam.com/. Last accessed 06 June 2024.

62. TensorFlow, 2024. URL https://www.tensorflow.org/. Last accessed 06 June 2024.

63. MathWorks - Makers of MATLAB and Simulink, 2024. URL https://www.mathworks.com. Last accessed 06 June 2024.

64. SciPy: Fundamental Algorithms for Scientific Computing in Python, 2024. URL https://scipy.org/. Last accessed 06 June 2024.

65. PyTorch, 2024. URL https://pytorch.org/. Last accessed 06 June 2024.

66. Talasila P, Mikkelsen PH, Gil S et al. Realising digital twins. In *The Engineering of Digital Twins*. Springer, 2024. pp. 225–256.

67. Feng H, Gomes C, Thule C et al. The Incubator Case Study for Digital Twin Engineering. Technical report, 2021. DOI: 10.48550/ARXIV.2102.10390. 2102.10390.

68. ISO. *Automation systems and integration - Digital twin framework for manufacturing*. ISO 23247:2021(E) ed. Geneva, Switzerland: International Organization for Standardization, 2021. URL https://www.iso.org/standard/78743.html.

69. Kamburjan E and Johnsen EB. Knowledge Structures Over Simulation Units. In *ANNSIM*. IEEE, pp. 78–89.

70. HUBCAP, 2024. URL https://www.hubcap.eu/. Last accessed 06 June 2024.

71. Cimatti A, Tian C and Tonetta S. Assumption-based runtime verification of infinite-state systems. In *Runtime Verification: 21st International Conference, RV 2021, Virtual Event, October 11–14, 2021, Proceedings 21*. Springer, pp. 207–227.

72. Kiss T, Kacsuk P, Kovacs J et al. Micado—microservice-based cloud application-level dynamic orchestrator. *Future Generation Computer Systems* 2019; 94: 937–946.

73. CP-SENS - Cyber-Physical Sensing for Machinery and Structures, 2024. URL https://digit.au.dk/research-projects/cp-sens. Last accessed 06 June 2024.

74. The C4 model for visualising software architecture, 2024. URL https://c4model.com/. Last accessed 06 June 2024.

75. Huang H and Xu X. Advancing digital twin implementation using edge adapters based on containerization. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*. IEEE, pp. 1–6.

76. Larsen PG, Talasila P and Fitzgerald J. Towards the composition of digital twins. In *The Application of Formal Methods: Essays Dedicated to Jim Woodcock on the Occasion of His Retirement*. Springer, 2024. pp. 103–122.

77. Böttjer T, Tola D, Kakavandi F et al. A review of unit level digital twin applications in the manufacturing industry. *CIRP Journal of Manufacturing Science and Technology* 2023; : 162–189DOI:https://doi.org/10.1016/j.cirpj.2023.06.011.

78. Digital Twin as a Service CLI, 2024. URL https://pypi.org/project/dtaas/. Last accessed 06 June 2024.

79. Jensen AM, Schoerghofer-Queiroz A, Ulriksen MD et al. Digital twin as a service for damage prognosis of offshore

wind turbine foundations. In *2024 International Conference on Noise and Vibration Engineering*.

80. InfluxDB Time Series Data Platform — InfluxData, 2024. URL https://www.influxdata.com/. Last accessed 06 June 2024.

81. MongoDB: The Developer Data Platform — MongoDB, 2024. URL https://www.mongodb.com/. Last accessed 06 June 2024.

82. RabbitMQ: One broker to queue them all — RabbitMQ, 2024. URL https://www.rabbitmq.com/. Last accessed 06 June 2024.

83. Eclipse Mosquitto, 2024. URL https://mosquitto.org/. Last accessed 06 June 2024.

84. OPC Foundation, 2024. URL https://opcfoundation.org/. Last accessed 06 June 2024.

85. Examples to demonstrate the features of DTaaS software, 2024. URL https://github.com/INTO-CPS-Association/DTaaS-examples. Last accessed 06 June 2024.

86. Grafana: The open observability platform, 2024. URL https://grafana.com/. Last accessed 06 June 2024.

87. YAML Ain't Markup Language, 2024. URL http://yaml.org. Last accessed 06 June 2024.

88. Tekinerdogan B, Blouin D, Vangheluwe H et al. (eds.) *Multi-Paradigm Modelling Approaches for Cyber-Physical Systems*. Elsevier, 2020.

89. Junghanns A, Blochwitz T, Bertsch C et al. The Functional Mock-up Interface 3.0 - New Features Enabling New Applications. In *Proceedings of the 14th International Modelica Conference*. online: Linköping University Electronic Press, Linköpings Universitet. DOI:10.3384/ecp2118117.

90. Gmsh: A three-dimensional finite element mesh generator with built-in pre- and post-processing facilities, 2024. URL https://gmsh.info/. Last accessed 23 September 2024.

91. Eclipse Vorto, 2024. URL http://yaml.org. Last accessed 06 June 2024.

92. Gil S, Oakes BJ, Gomes C et al. Toward a systematic reporting framework for digital twins: a cooperative robotics case study. *SIMULATION* 2024; : 1–27DOI:10.1177/00375497241261406. URL https://doi.org/10.1177/00375497241261406.

93. Frasheri M, Ejersbo H, Thule C et al. RMQFMU: Bridging the Real World with Co-simulation For Practitioners. In Macedo HD, Thule C and Pierce K (eds.) *Proceedings of the 19th International Overture Workshop*. Overture.

94. Feng H, Gomes C, Sandberg M et al. Developing a physical and digital twin: An example process model. In *2021 ACM/IEEE International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*. IEEE. DOI:10.1109/models-c53483.2021.00050.

95. TeSSLa Telegraf Connector, 2024. URL https://tessla.io/blog/telegrafConnector/. Last accessed 06 June 2024.

96. Lumer-Klabbers G, Hausted JO, Kvistgaard JL et al. Towards a digital twin framework for autonomous robots. In *2021 IEEE 45th annual computers, software, and applications conference (COMPSAC)*. IEEE, pp. 1254–1259.

97. Gil S, Mikkelsen PH, Tola D et al. A modeling approach for composed digital twins in cooperative systems. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, pp. 1–8.

98. Gomes C, Thule C, Broman D et al. Co-simulation: a survey. *ACM Computing Surveys (CSUR)* 2018; 51(3): 1–33.

99. Thule C, Lausdahl K, Gomes C et al. Maestro: The into-cps co-simulation framework. *Simulation Modelling Practice and Theory* 2019; 92: 45–61.

100. Larsen PG, Esterle L, Fitzgerald J et al. Fault Injection in Co-simulation and Digital Twins for Cyber-Physical Robotic Systems. In *Applicable Formal Methods for Safe Industrial Products: Essays Dedicated to Jan Peleska on the Occasion of His 65th Birthday*. Springer, 2023. pp. 222–236.

101. Ejersbo H, Lausdahl K, Frasheri M et al. FmiSwap: Run-time Swapping of Models for Co-simulation and Digital Twins, 2023. 2304.07328.

102. iPerf - The ultimate speed test tool for TCP, UDP and SCTP, 2024. URL https://iperf.fr/. Last accessed 06 June 2024.