# Obtaining services from Digital Twins in industrial settings

Anonymous

**Abstract** Digital Twin (DT) technologies are envisioned to bring substantial benefits to industry by optimising production processes, thus reducing the costs and time-to-market of products. A DT is a digital representation of a real-world entity which we call the Physical Twin (PT). The DT and PT are connected by a communications infrastructure which allows the DT to maintain a known level of fidelity to the PT it represents. A DT leverages the models at its core to offer its stakeholders a range of services that add value to the PT without unduly compromising the PT's operation. These services could include visualisation and monitoring, predictive maintenance, fault diagnosis, what-if analysis and runtime reconfiguration of the PT. The key concepts behind DTs, as well as ways to obtain services from them in industrial settings, coupled with reports on existing case-studies, are the main topics covered in this Chapter.

## 1 Introduction

Over the years many companies have integrated software components in their physical systems for better monitoring and control, resulting in Cyber-Physical Systems (CPSs) (see Figure 1). The success of CPSs has shown not only the benefit of integrating software in physical systems to the initial purpose of the system but also the potential of the data generated by the CPSs to provide additional value. However, the contexts and environments in which such CPSs operate may change during deployment and hence theirr performance degrades. Extending software components with models of the system and its environment and introducing sensing capabilities both within the system and its environment allows to create a Digital Twin (DT) of the CPS. Such DT can be used to monitor the CPS and its environment and to predict the future behaviour of the CPS. Thus, it makes sense to consider systematically supporting such CPSs with a specific DT. In addition to monitoring and prediction of the systems performance and behaviour, DTs can be equipped with different services that can increase the value that the CPS provides for its user. In this chapter we elaborate how such services can be established in an industrial setting.

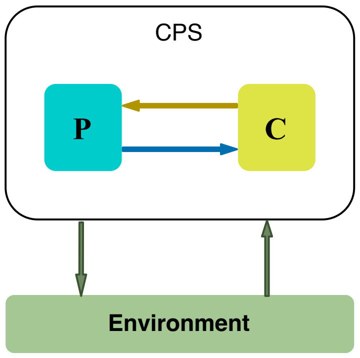


**Fig. 1** The structure of a CPS (adapted from [1]).

CPSs that are supported by a DT are considered as Physical Twins (PTs) and jointly this is called a DT-enabled system. It is important that the DT provides additional support and gives additional value compared to the CPS, its PT, itself. In an industrial setting the CPSs can be considered as business-to-business assets. In this fashion they form an eco-system where there typically are clear rules for sharing information between the organisation using the CPS and the manufacturer of the CPS. In such cases, the manufacturer is able to provide services on top of the CPS. These services can support the operation of the CPS during its lifetime and the data from the CPS can typically give value for both users and manufacturers. For the manufacturers the data can in particular give value about how the CPS is being used, which is valuable for the engineering of new products with a functionality similar to the CPS, or when maintenance of the CPS is required, reducing downtime and unforeseen operational halts. In addition, manufacturers may be able to provide services that they can charge their customers for if (and only if) they are able to provide the additional value of these services for the users.

The rest of this chapter presents a proper definition of the characteristics of DTs in Section 2. Afterwards, the different techniques needed to construct DTs are explained (Section 3). This is followed by Section 4 on the different kinds of services that can be provided by a DT. Finally, the chapter is completed by Section 5 with the future directions for DTs and Section 6 for concluding remarks.

## 2 What are Digital Twins

DTs have originally emerged as a concept in the works of Grieves and Vickers, who focused on the management of product life-cycles [2]. Several applications have been inspired from the initial idea, serving to investigate biological systems on one side [3], and sustainable development on the other [4]. Moreover, DTs are gaining traction and are being adopted in manufacturing [5], automotive industry [6], energy [7, 8], agriculture [9], as well as national infrastructures[1].

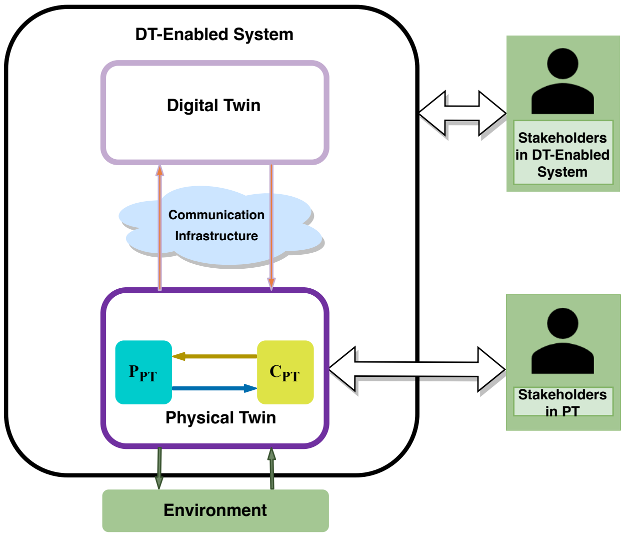


**Fig. 2** Overview of a DT-enabled system. Stakeholders can be involved both in the PT and DT (adapted from [1]).

It is possible to find different definitions for DTs, the reader is pointed to [23, 32, 35]. In this chapter, the following definition is adopted given by [10], where:

## DT Definition

A DT is a digital representation of a real-world entity which we call the Physical Twin (PT). The DT and PT are connected by a communications infrastructure which allows the DT to maintain a known level of fidelity to the PT it represents. A DT offers its stakeholders a range of services that add value to the PT without unduly compromising the PT's operation.

It becomes apparent from this definition, that the DT needs to correspond to some PT of interest, with which there is (bi-directional) communication at runtime; we refer to this constellation as a DT-enabled system. Additionally, the DT can be seen as a collection of various assets, such as models, that capture relevant aspects of the PT. It is further understood that these models should come with certain fidelity, such

[1] https://www.cdbb.cam.ac.uk/DFTG/GeminiPrinciples

that the services provided by the DT can have practical use. The services themselves are to reflect the desires from stakeholders, e.g., preventive maintenance, failure detection and recovery, to name a few, without undermining the PT in fulfilling its own demands.

A DT would typically consist of (i) models, e.g., first principles, data-driven, etc., (ii) data, which can be communicated or sensed directly from the environment, and (iii) services, e.g., visualisation, monitoring, what-if analysis etc., see Figure 3. Note that, different stakeholders will desire to interact with the DT at different levels (depending upon their role).

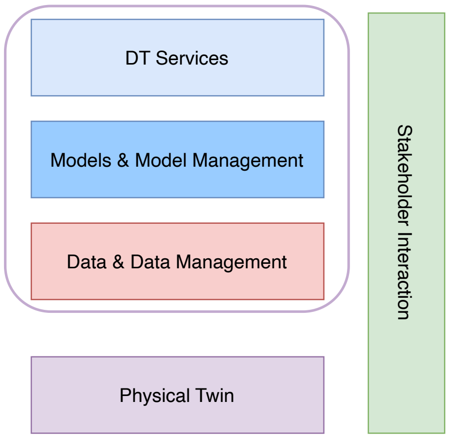


**Fig. 3** Peeking inside the Digital Twin (adapted from [1]).

## 3 Constructing Digital Twins

DTs can be built by the meaningful integration of relevant models, data, and the exchange thereof with the corresponding PT, and services which add value to the interested stakeholders. Services are covered in-depth in the next section, exemplified through different case-studies. This section focuses on the models and data at first. Thereafter it provides a brief overview of a DT platform, namely the DT as a Service (DTaaS) platform, which provides support to users to built DTs in reusable way, thus enabling both users and developers to share DTs, or specific DT components between them.

## 3.1 Models

DT models consist in digital representations of relevant aspects of the PT. The goal of the model is not to capture every detail of what it is a representation of. Instead, the model covers a subset of properties of its system, and can be used in its place in well-defined scenarios. The model can be used to experiment and simulate without running the real system, in order to gain insights into the system itself. It is therefore paramount to speak of the fidelity of the (any) models, and providing bounds pertaining the errors between models and reality.

There is a variety of modelling formalisms from which DT engineers can choose from. The ones relevant in a DT context can be grouped in three categories: physics-based, data-driven, and computer-based models. Physics-based models are built on first principles and take the form algebraic, differential equations, or a mix of the two. Differential equations enable the introduction of time in algebraic equations, with differential inclusion covering those systems that abide to probability distributions. Ordinary differential equations or partial differential equations can be used. The former use basic conservation laws (energy, momentum) for the formulation, and the solution is computed through time integration algorithms. The latter also use the basic conservation laws but are instead used for capturing infinitesimal portions of a physical systems. Such models also consider geometrical representations of the physical system. The analytical solutions to these problems when available covers only cases with simple geometry. For complex systems, numerical approximations are used instead. An example of a physics-based model is provided in Figure 4.

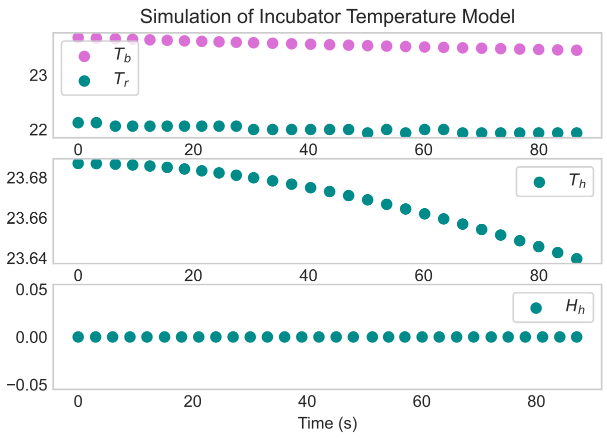


**Fig. 4** Modeling the temperature for an incubator system using time integration for 30 steps, where $T_h$ is the temperature of the heater, $T_b$ is the temperature inside the incubator, $T_R$ is the room temperature, and $H_h$ is a Boolean variable denoting an electric switch (on/off) (adapted from [1]).

Data-driven models uses data to learn the model, that is the input-output relationship. These models target systems that are quite complex, the behaviour of which might not be accurately represented by physics models solely. The basic example of such a model is linear regression. Machine Learning (ML) techniques takes regression further and makes it applicable to problems of greater complexity. Artificial neural networks are a well-known data-driven formalism (see Figure 5 for the classic neuron model). In addition, some researchers have started to add physics information to their ML models, thus building physics-informed ML [11]. Empirical data coming from the system is used in order to learn relationships and patterns that help capture the behaviour of the system. In other words, what is being learnt are model parameters. An obvious challenge for building such models is the availability of large amounts of data, as well as quality data to ensure the accuracy of the model. The latter relies heavily on the purpose of use of the model. Moreover, real-time requirements should also be met in a DT setting.

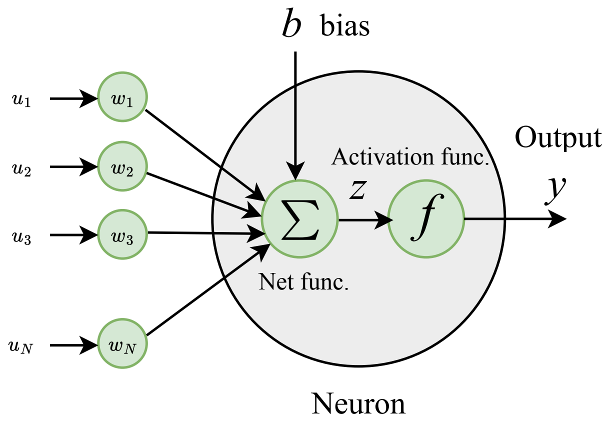


**Fig. 5** Classic neuron model from McCulloch and Pitts [12].

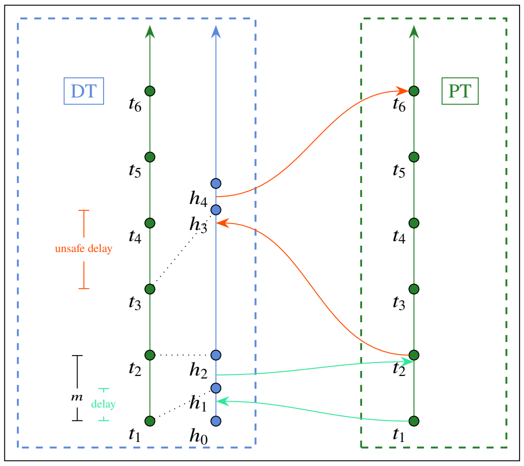
Computer-based models target the behaviour of hardware and software components of a computers systems. Discrete logic can be used to describe these systems, as well graphical and textual specifications can be developed. Different tools are available for this type of modelling. Finite state machines allow the modelling of system behaviours through discrete states connected by transitions, and rules that prescribe when the transition between states should take place [13]. The Vienna Development Method (VDM) is an established textual formalism for modeling computer systems based on specifications [14, 15, 16]. Verification method capture system behaviour in a contractual manner, specifying the pre and post conditions of the execution of some part of the system. The behaviour can then be verified either through reasoning techniques using formal logic, or through exhaustive methods, exploring all possible executions and checking if the specifications hold. CPSs however are quite complex systems and generally require multiple formalisms for modelling. Co-simulation techniques are used instead [17], which allow engineers to couple these models in a co-joint simulation to verify and explore a system's behaviour. To this end, standards are needed to make such coupling possible. The Functional Mock-up Interface (FMI) is one such standard [18, 19, 20], well established in academia and industry that prescribes how the models should be packaged and how the interfaces should be specified.

## 3.2 Data

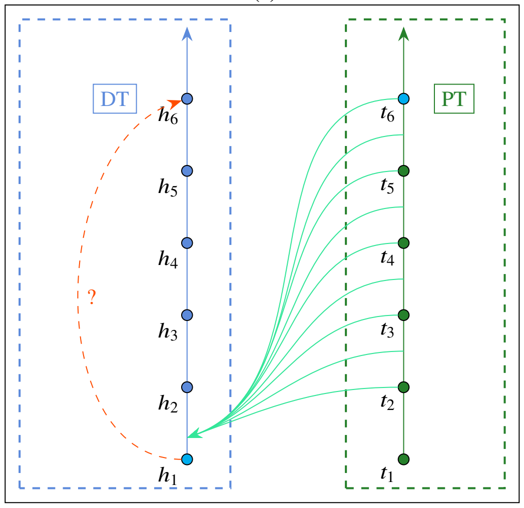
The communication between DT and PT is crucial in DT-enabled systems. The real data coming from the PT is used to supplement the experimentation and simulation based on the models, allowing DT services such as fault diagnosis or preventive maintenance, to name a few. The communication infrastructure needs to be in place such that the data can flow from the PT to the DT. Said data might not be directly usable in its raw state, but could need further processing in the DT. In addition, the DT needs to store and manage the data, supporting also the possibility that it is in the form of a time series. Data coming from different PT sensors would also need to be fused in a meaningful way to be usable.

A few considerations are to be made on the communication. The data coming from the sensors is limited by their sampling frequency, and depends on the type of analog-to-digital converters used. Furthermore, even if the sampling frequency can be deemed accurate in a particular case, the communication between DT and PT will be affected by delays and jitter, thus impacting the time when the samples are received on the DT side. The synchronisation mechanisms between DT and PT should be accounted for and will depend on the approach that is taken to implement the DT. Network drops and degradation have been considered in the work of. Frasheri et. al [21], where the DT is implemented as a co-simulation, coupled with the PT through for example the RabbitMQ protocol. In their system, the DT clock move forward upon recieval of a message from the PT. In this setup, time discrepancies can arise between PT and DT should there be delays in the communication. This point is illustrated in Figure 6. It is clear that when the DT and PT are out of sync, care should be taken for both sides to be aware of this occurrence to ensure that obsolete messages do not affect system behaviours.

It is also worth mentioning that, as the DT can affect the PT, the characteristics of the PT signal may be subject to change. During the initialisation of the DT, its view of the world could be skewed, and as such should be accounted for by correcting relevant parameters.



(a)



(b)

**Fig. 6** a) Network degradation scenario resulting in higher delays. Green arrows indicate message going through within the expected delays; red arrows indicate transmissions during network disturbances. $t_i$ is the wall-clock time, (assumed) shared for the DT and PT; $m$ is the age constraint. Dotted lines represent the correspondence between DT and PT times. b) Message burst after connection becomes available again. The current time-step (simulation and wall-clock time) is depicted as a circle filled in cyan. The dashed red arrow indicates the possibility to go directly from $h_1$ to $h_6$, as opposed to taking steps one-by-one, i.e. $h_1$, $h_2$, ..., $h_6$. Figure based on [21].

## 3.3 DTaaS Platform

As the interested in DT grows, so do DT frameworks that use Internet of Things (IoT) technology to support DT services, both on the open-source and commercial fronts. The reader is pointed to a recent survey in the field [22]. Nevertheless many such frameworks target specific application domains, and do not provide support for different services such as visualization, state estimation, monitoring, and anomaly detection to the level of the DTaaS platfrom.

**Table 1** Requirements for building a DT platform such as the DTaaS.

| **Role** | **Requirements** |
|---|---|
| PT design | 1. Access to historic data from PTs in their operational environment.<br>2. Create PT prototypes from available PT components.<br>3. Create reusable assets for the DTs and PTs. |
| PT manufacturing | 1. Save test data from the available PT prototypes.<br>2. Connect PTs to prototype DTs running on the DT platform.<br>3. Tweak DT and PT prototypes to support what-if analysis. |
| Model management | 1. Create and maintain complementary DT models.<br>2. Enable composability to support system of sysstems.<br>3. Test DTs with various models to evaluate the adequacy of a model for a PT-DT combination. |
| PT maintenance | 1. Utilise DT services to gain insights on the PT state, current and past.<br>2. Enrich the DT with monitoring, predictive maintenance services.<br>3. Re-configure the PT and DT system based on the recommendations from the DT services. |
| DT architecture | 1. Enable the composition and configuration of the DT and other required services.<br>2. Enable the analysis and decision making on one DT, or a fleet of DTs.<br>3. Integrate DTs with additional internal and external services that are available to satisfy other requirements. |
| Calibration | 1. Configure/reconfigure the DT as a result of the state of PT and its environment.<br>2. Create (potentially) reusable calibration services for DTs. |

The DTaaS platform was proposed to enable users to build, use, and share their DTs, thus promoting reusability of the different DT assets (models, data, or services) as a whole or asset-wise. The requirements for building such a platform are captured in Table 1.

The architecture of the platform, a simplified view of it, is depicted in Figure 7. The platform is composed of a series of containers to support scalability, each fulfilling one or more of the requirements presented above.

The data processing and storage container brings together technologies that are data related, such as databases or data lakes. Technologies used to process PT data such as Apache Spark, the InfluxDB could be included in this container. Secure access to these services is provided.

The management of assets is carried out in the reusable assets container, where functionalities like version control, traceability are supported. A variety of tools are available for creating DT assets, especially models.

Dedicated containers are available for providing high-level services to users, e.g., visualisation and monitoring, calibration and reconfiguration. Jupyter notebooks could quite useful for example for supporting (interactive) visualization. Users are also supported in adding their own services. Note, that is not only models and data that can be reused across users, but services as well.

The DT lifecycle manager handles the lifecycle of the DT through its phases, from creation, to deployment, maintenance, and termination. The DT execution manager on the other hand makes use of the configuration of the DT, and executes the latter in the cloud and corresponding virtual environments. Security is also provided to the user through the authentication component.

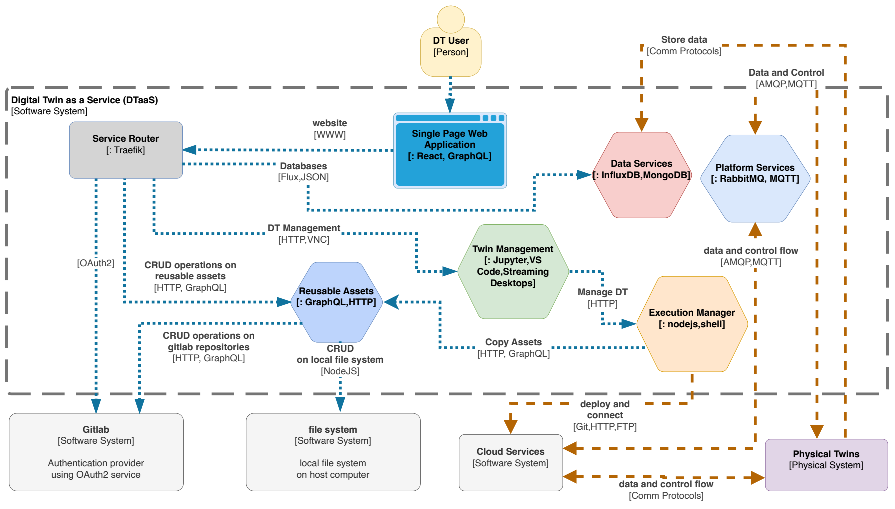

DT User [Person]
Store data [Comm Protocols]
Data and Control [AMQP,MQTT]
Digital Twin as a Service (DTaaS) [Software System]
website [WWW]
Service Router [: Traefik]
Single Page Web Application [: React, GraphQL]
Databases [Flux,JSON]
Data Services [: InfluxDB,MongoDB]
Platform Services [: RabbitMQ, MQTT]
DT Management [HTTP,VNC]
data and control flow [AMQP,MQTT]
Twin Management [: Jupyter,VS Code,Streaming Desktops]
[OAuth2]
CRUD operations on reusable assets [HTTP, GraphQL]
Reusable Assets [: GraphQL,HTTP]
Manage DT [HTTP]
Execution Manager [: nodejs,shell]
CRUD operations on gitlab repositories [HTTP, GraphQL]
CRUD on local file system [NodeJS]
Copy Assets [HTTP, GraphQL]
deploy and connect [Git,HTTP,FTP]
Gitlab [Software System] Authentication provider using OAuth2 service
file system [Software System] local file system on host computer
Cloud Services [Software System]
Physical Twins [Physical System]
data and control flow [Comm Protocols]

**Fig. 7** Simplified architecture of the DTaaS platform (adapted from [1]).

## 4 Digital Twin Services

The services provided by the DT rely heavily on the available models and data, and can be grouped in three categories, corresponding to visualisation, monitoring, and decision support. To illustrate these services with concrete examples, the case-study of a simple incubator is adopted [23], the goal of which is to maintain the temperature inside its box at a specified value.

### 4.1 The Incubator Case-study

The incubator can be considered as a simpler version of industrial examples (dry ovens, organ preservation boxes [24]), which retains some of the features of interest without the full complexity. In our case, the incubator is used for the production of tempeh, a fermented food popular in Indonesia and in some other parts of Southeast Asia [25] (Figure 8). For this purpose, in order to enable fermentation, the temperature within the incubator has to be kept around 37.5 °C.

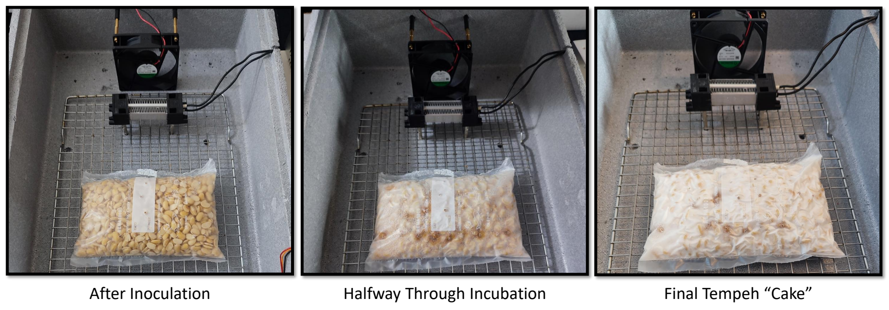


**Fig. 8** Phases in the fermentation process: soaked soy beans at the start (left), half way through the incubation the femerntation has started (middle), final product (right). (Taken from [1].)

The incubator consists of an insulated box, inside of which are located a heatbed, temperature sensors, and a fan (as shown In Figures 9 and 10). Outside of the incubator there is another temperature sensor to measure the external temperature. The standard controller is a simple ”bang bang” controller, which switches on and off the heater and the fan to ensure that the temperature is kept at the desired level.

The rationale behind adopting a DT solution for such a system is summarized as follows. First, the access of historical data would be made available to the user supporting the detection of anomalies in the fermentation process, thus enabling reconfiguration. Second, conserving energy in the face of potentially inadequate user actions, such as leaving the lid of the incubator open. Third, providing a notification to the user when the product is ready, thus minimizing the need for disturbing the

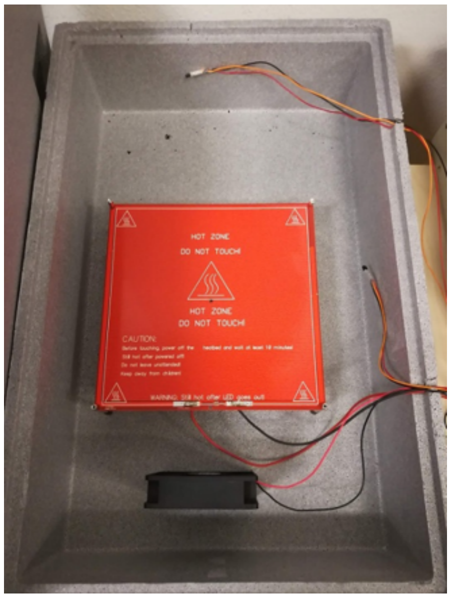


**Fig. 9** A view inside the incubator. (Taken from [1].)

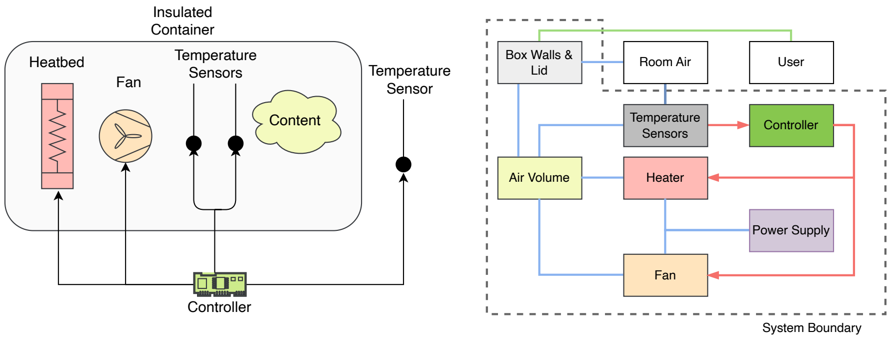


**Fig. 10** Schematic overview of the incubator system (left), and its block diagram (right). Blue lines are energy connections, red lines indicate the flow of digital information, and green lines correspond to user actions (adapted from [1]).

process during fermentation. Fourth, notifying the user when maintenance actions need to be carried out. Note that, the incubator without a DT can still meet its fundamental requirement of maintaining the desired temperature. A DT-enabled incubator would be supported by additional services that optimize performance, e.g., energy consumption, when possible. The following subsections describe the possible different services in detail.

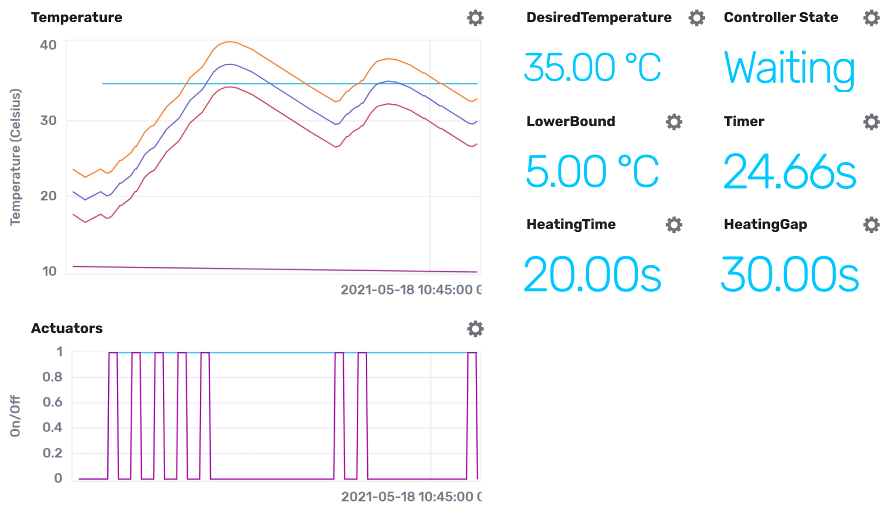
## 4.2 Visualisation Services

Visualisation would enable a user to get an overview of the current state of affairs within the PT (see Figure 11). It could also support the inspection of historical data, as well as display such data alongside the results of different types of analyses conducted, in order to guide decision-making. This can be imagined as a web server, to which a user connects to and accesses dashboards and live plots of relevant parameters.

The implementation of visualization interfaces should take in consideration the overall purpose, i.e., what should be visualized, and at what dimension the information would be portrayed. This is not a trivial problem, and research within this area typically falls under Human Computer Interaction (HCI), that deals precisely with the optimisation of such interaction, such that it is both a positive experience for humans and that the information is conveyed clearly. Moreover, in the contexts of virtual environments, Virtual Reality (VR), and Augmented Reality (AR), these interactions tend to be quite different to the traditional interfaces [26]. Furthermore, it could be quite useful in improving the experience of operators interacting with CPSs and DTs throughout the lifecycle of such systems, e.g., for guidance and general information, as well as training (manual assembly) [27]. Consider for example users that are quite curious and thus prone to interacting with the system even if such interaction would interfere with normal operations. An AR visualisation option can allow a user to play with, and learn about a system, without tampering with the real PT itself. Moreover, during the operation of the PT, there might a need for manual intervention. In this case, AR can provide users an immersive guide on what to do step-by-step.

While there are many visualization techniques available, only five are covered in this Chapter to serve as a starting point, as follows:

1. Data graphs and charts are quite useful for displaying data, making it possible to see changes over time, and potentially identify trends. This is especially the case for systems that generate large amounts of data. An example of this technique is shown in Figure 11.
2. Schematic diagrams in 2D that change over time together with the system that they represent could be useful to show how the interactions between the system's components change over time. In addition they could help in identifying which parts could be a cause for issues. These diagrams are typically floor plans or 2D maps.
3. 3D models showing different physical characteristics of the PT, such as its shape, size, and internal components, could also be useful, especially if interactive allowing a user to rotate, zoom, or change perspectives of the model. A greater detail could also be achieved with 4D if so desired.
4. Augmented reality enables the addition of digital information over the physical world representation in a DT. As such it is quite helpful as it provides users simultaneous access to both physical and digital characteristics of the PT they want to inspect.



35.00 °C
Waiting
5.00 °C
24.66s
20.00s
30.00s

**Fig. 11** Visualisation example with InfluxDB for the Incubator. Taken from [1].

5. Mixed reality makes use of interactive control, e.g., haptic feedback, to provide a 4D experience to users. This technique is useful for those users who need high precision and control in their DT.

Naturally, it is not needed to have fully blown 4D environments for every system. In the case of the incubator even a 3D model might have limited use, as the incubator system is static, unlike for example a robotic arm, where the simulation of the robots motion is crucial. Different frameworks are available for supporting visualization services in a DT, such as dashboards and 4D environments.

A few of the frameworks mentioned in this Chapter have been designed with IoT and cloud connections in mind, such as InfluxDB, JMobile, and Grafana. InfluxDB is a cloud-based-time-series database, with APIs in C#, Java, and Javascript. Data can be stored and visualized in customizable way. JMobile supports process management, data visualization, as well as supporting data exchange between the edge and cloud. It supports several protocols like PROFINET, EtherCAT, EtherNet/IP, Modbus, and SQL. Grafana is another web-based alternative, with which it is possible to visualize data through dashboards either in the cloud or locally.

Regarding 4D environments, several options are available. The Unreal Engine and Unity are both computer graphics engines, initially intended for the gaming community. Recent developments have these platforms being used in other domains such as automotive, transport, mobile applications etc.

## 4.3 Monitoring

Monitoring consists in observing selected parameters and behaviours of the PT during operation, in order to raise an alarm in case of anomalies and undesirable situations. In general, monitoring is realized using either model-based or data-driven techniques.

Model-based monitoring consists in the observation of the PT's behaviour and checking whether such behaviour is expected or if it violates some pre-defined conditions. The behaviour of the PT is expressed through property specifications over the states of the PT and its environment, using some logic language, e.g., Temporal Logic (TL) [28, 29]. A sequence of states is usually called a trace.

Properties could be specified using simple propositions, e.g., "the temperature in the incubator is 30 °C", and can be true or false. To account for the behaviour of the PT over an extended period of time, and capture scenarios where the temperature might be different at a later time, it is possible to use predicates, supported in first-order TL. There are logic frameworks beside TL that can be adopted. It is not a goal of this Chapter to go into details, as such the reader is pointed to [30].

In Figure 12 it is possible to see an example of a DT that monitors a given PT. Data flows from the PT to the DT, and is used to generate traces. Traces are thereafter fed to the monitor which computes a verdict on whether the property specifications are satisfied, e.g. "the temperature of the incubator is at all times under 40 °C". This verdict could be used to alert a human operator in case of violations, or directly control the PT, e.g., switching off the heat-bed for a period of time.

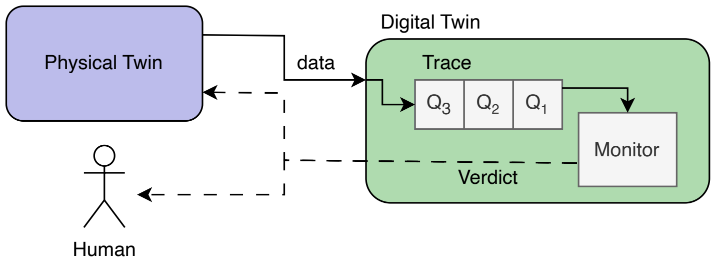


**Fig. 12** Model-based monitoring as a DT service (adapted from [1]).

Specifically, desirable properties could be specified over these properties to ensure the PT complies with what is expected, and to raise the alarm when it does not as is the case with anomaly detection. In the case of the incubator such an example would be, monitor the temperature inside the box such that it does not go over 40 °C. Note that in regards to DTs, monitoring is performed online, and the model-based approach to it as described above corresponds to runtime verification [31]. In addition, since the data from which the traces are being generated is coming from the PT, communication requirements needs to be specified. It should also be ensured that the monitoring is minimally intrusive to the operation of the PT.

Data-driven approaches on the other hand make use of the data coming from the PT to learn parameters of anomaly models that correspond to different types of faulty operations. Moreover, this is often treated as a one-class classification problem (e.g. in [32]), focusing on learning what normal operation looks like. The lack of data for anomalous situations is the main reason for this. Additional datasets for such conditions would need to be created, which is a costly and time-consuming process.

Assume in the case of the incubator that data has been collected corresponding to its normal operation. Furthermore, note that each (temperature) measurement corresponds to one particular sensor at some specific time. These measurements are not independent, in fact they form a time series. In this context, a data-driven model, or an ML model can be used to detect normal and anomalous operation, which takes as input the time series, considering both static measures (temperature), and dynamic ones (rate of temperature change).

Anomaly detection can be performed at different levels, both at the individual component/sensor level and the system/incubator level. To tackle the former, data from the sensor can be used to train the model. Should there be more than one sensor, depending on how much data is available, one can choose to perform anomaly detection for each sensor (more accurate), or combine the data over the multiple sensors. Another alternative would be to use data reconstruction techniques, e.g. Autoencoder networks. These have low error for reconstructing normal operation inputs, and high error for anomalous ones. In the case of system level anomaly detection, data from all sensors and components is combined, in other words the time-series corresponding to each component are concatenated into a multi-dimensional time-series. The latter are thereafter fed to recurrent neural networks which do recosntruction or prediction in a similar way to the component level anomaly detection.

## 4.4 What-if Analysis

What-if analysis is a technique that allows the exploration of a series of scenarios in simulation by sweeping ranges of different parameters, as well as using historical data to provide insights in regards to the PT, e.g., in terms of finding an optimal value or strategy to deploy in a specific scenario. Such analysis is rather heavy in terms of data and computation. Moreover, for the results to be useful, simulations should be carried out faster than real-time, and an answer from the analysis should come in a timely fashion.

Consider as example that there is a need to substitute the power supply in the incubator without stopping operation, in this case the fermenting process. A strategy would be to boost the temperature prior the replacement, hoping that the temperature remains in range. The length of this boost, reffered to as H, could be evaluated through a what-if analysis. The outcome of such a simulation is depicted in Figure 13.

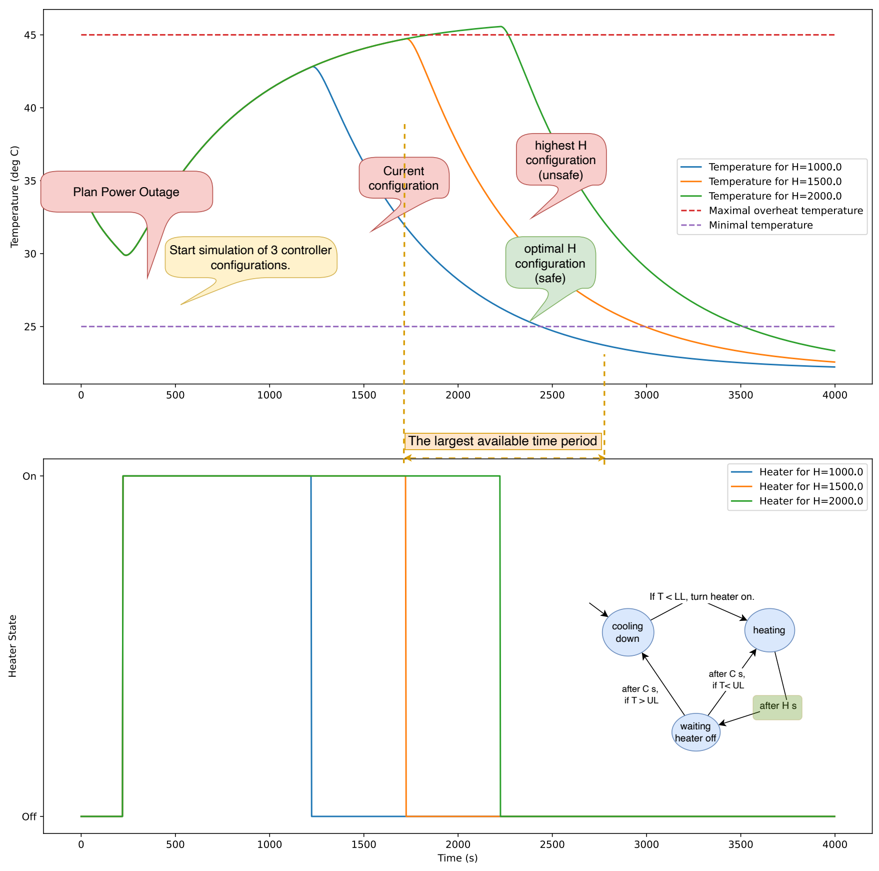


**Fig. 13** Results of what-if simulations for the incubator (adapted from [1]).

Design Space Exploration (DSE) is another technique that has a similar working principle to what-if simulations, in that a series of simulation are run, where ranges of parameters of interest are swept through [33]. There is however a fundamental difference between the two, DSE is used during the design phase of a system, e.g., for finding the optimal position of the fan, whereas what-if simulation refer to the operational stage, thus simulating the execution of the system.

Genetic algorithms can be used when performing DSE in order to select what simulations to run next, to more effectively sweep the design space. Especially when the latter is too large, and simulating every single scenario is not feasible, at least not in reasonable time for the results to be useful. It can also be very common for there to be conflicting objectives when specifying the optimization goals. Ranking functions, or Pareto Fronts [34] could be used to find the “best” design.

An example of a DSE for the incubator is presented in Figure 10. Two conflicting objectives are considered in this case, namely the number of times the heater is turned on and off (actuator effort), and the integral of the temperature in comparison to the desired one (the controller error). The trade-off is quite apparent in Figure 14, and can be seen in the resulting Pareto Front.

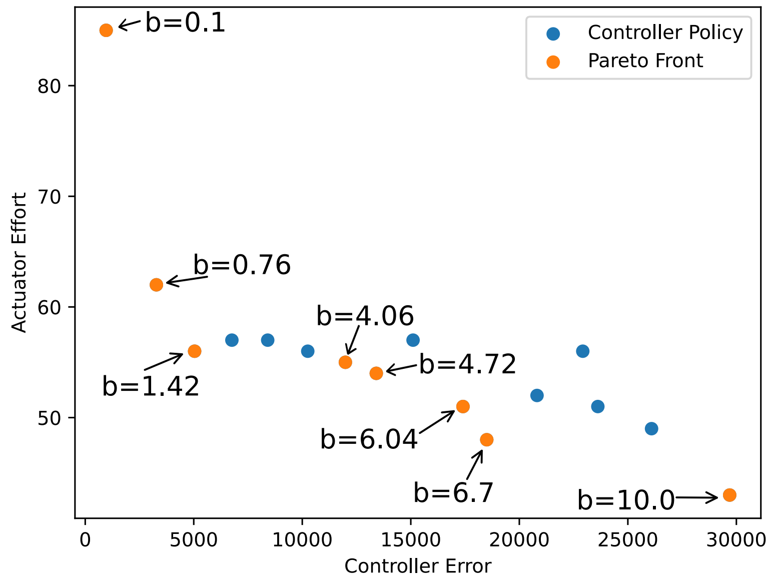


**Fig. 14** DSE results for the incubator with two conflicting objectives, where *b* is a controller parameter for which it is being optimised (adapted from [1]).

What-if analysis could also be coupled with Fault Injection (FI) techniques to find which strategies are most suitable in the presence of different faults, and potential system failures [35]. In this manner, potential consequences can be evaluated before a strategy is deployed to the PT.

FI techniques have been used for years to validate system dependability and robustness [36]. Early work dates back the work by H. Mills in 1972 at IBM [37]. Later on, FI was widely considered both by academics and industry, to understand how systems are affected by faults, and more importantly how to build fault tolerance mechanisms such that systems can degrade gracefully when faults lead to system failures [37]. Note that faults can happen at the software and hardware levels, and FI techniques have been adopted at both levels as well.

Faults are inevitable, as such the enhancement of DTs with FI capabilities is crucial for evaluating their impact in a PT in a controlled manner, without the need to test the PT itself in extreme conditions. What-if simulations and FI can be combined to explore different scenarios, thereafter carrying to the field the most interesting ones, limiting costs of testing and possible damages to the PT.

An FI simulation was performed with the incubator (depicted in Figures 15 and 16). The intent was to simulate the effect of manually removing the lid from the incubator, thus interfering with the fermenting process. The models in the DT in this case end up using an incorrect value for the air temperature. A Kalman filter is also integrated which predicts the temperature, and its output is different from the real temperature value (Figure 15).

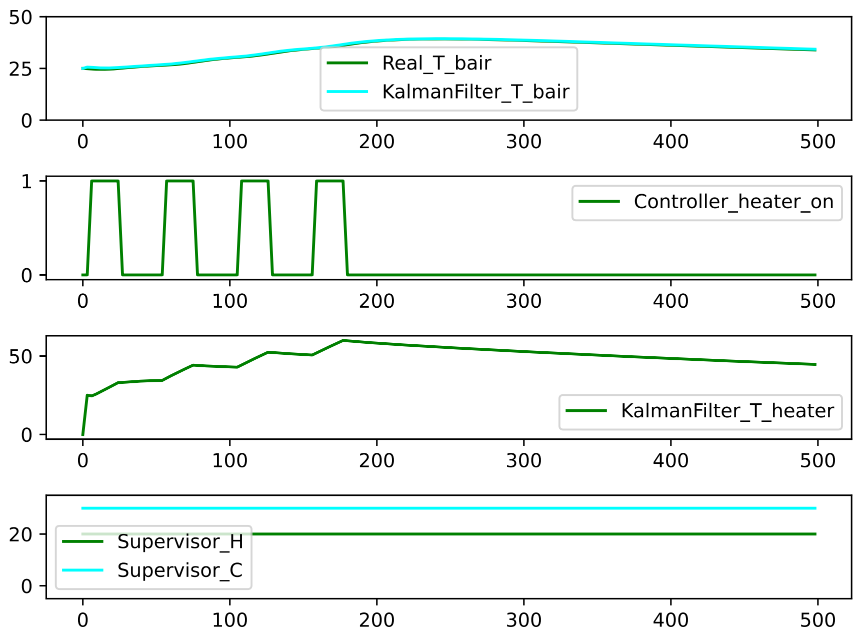


**Fig. 15** Simulation of the incubator without FI. X-axis simulated steps, y-axis temperature (plots 1,3,4) and on/off (plot 2). Adapted from [1].

## 4.5 Fault Diagnosis

Fault Diagnosis techniques consist of several steps starting with fault detection, isolation, and finally identification. Detection of faults is realized through monitors as already discussed in Section 4.3. On the other hand, isolation and identification deal with fault location, and other facets like type, shape, and size. Inputs to fault diagnosis mechanisms generally consist in fault symptoms, as well as available

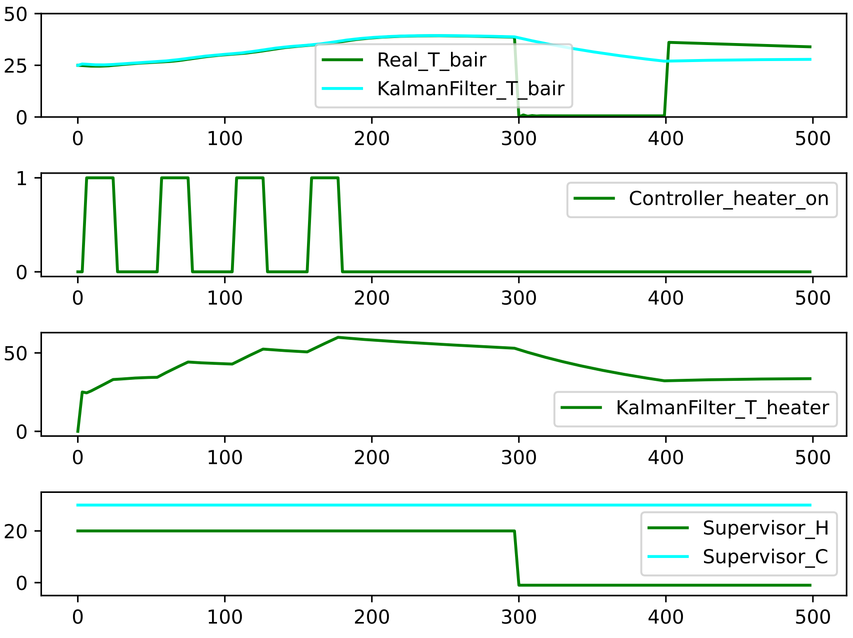


**Fig. 16** Simulation of the incubator with FI. X-axis simulated steps, y-axis temperature (plots 1,3,4) and on/off (plot 2). Adapted from [1].

heuristic knowledge on the fault [38]. Fault symptoms can amount to measured signals that are not within an acceptable, or expected range. Heuristics could be operator observations in terms of unexpected noises or visual inputs. Faults and symptoms have a causal relationship, and could be expressed with if-then rules. If these are not known before hand, classification methods can be trained and used instead.

Classification techniques for industrial CPSs typically fall under one the following categories: (i) physics-based and model-driven, e.g., Kalman filters, Markov models, fault trees etc., (ii) data-driven Artificial Intelligence (AI), e.g., neural networks, ML, fuzzy logic etc. and (iii) Knowledge-based, e.g., Bayesian networks, binary decision trees, etc. In the case of the incubator, one potential fault to be diagnosed is that of manual lid removal. The reader is pointed to [39] and [40] for examples, namely in assembly lines and smart manufacturing.

### 4.6 Predictive Maintenance

Predictive Maintenance consists in the analysis of long term trends with the goal of predicting and avoiding faults before they happen, by triggering maintenance actions in good time [41, 42]. These techniques require large amounts of data and rely on identifying patterns, trends and anomalies. They can also be combined with ML in order to identify early signs of the degradation of some component. The benefits of using predictive maintenance techniques are clearly visible. First, unexpected failures would be reduced, in turn leading to less downtime or other disruptions in the execution of the PT. Second, maintenance operation themselves are optimised, with unnecessary tasks being reduced, resulting in lower maintenance costs. Thirdly, the lifespan of the PT is increased due to the components being replaced or repaired in good time.

In the incubator example, components such as the fan can degrade over time, leading to higher motor friction, and to a higher current drawn. This can be observed in Figure 17. Sampling periodically such current could be used to calculate when the component will fail, as the more current that is drawn, the more the element is overheated.

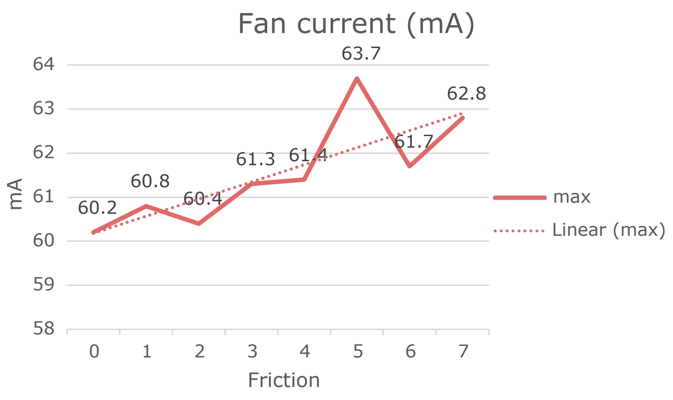


**Fig. 17** Drawn current from the fan as friction increases (adapted from [1]).

### 4.7 Re-configuration

Re-configuration aims to optimise the PT during operation, while minimizing downtime. In this way the DT can affect meaningful changes in the PT. The techniques discussed above, from monitoring to fault diagnosis, are the necessary ground work that serves as input to re-configuration processes. In other words, should an anomaly be detected by the monitors, the DT could utilize fault diagnosis techniques to isolate and identify faults. Thereafter, a what-if analysis could be performed to evaluate potential mitigation strategies.

A well known way to support re-configuration is the Monitor-Analyze-Plan-Execute over a shared Knowledge (MAPE-K) loop [43], which consists of four phases of self-adaptation as well as the knowledge component. A MAPE-K loop has been implemented in the incubator and evaluated in the context of one potential fault. Assume that someone removes the lid from the incubator during operation [44] (see Figure 18). As the PT is configured to work with a closed lid, by opening the lid the working models become invalidated, specifically the controller. The intervention of a human would be necessary to get the PT back on track. In order for the temperature to be maintained, the heatbed starts working even more. This is not desirable, as there is heat loss and a potential fire hazard. To prevent such occurrence the models need to be recalibrated. To solve the problem, first a Kalman filter is adopted for anomaly detection [23]. Thereafter re-calibration takes place, and parameters, specifically the maximum heating period, are re-calculated. In this case the latter is capped such that the heatbed does not work more than specified.

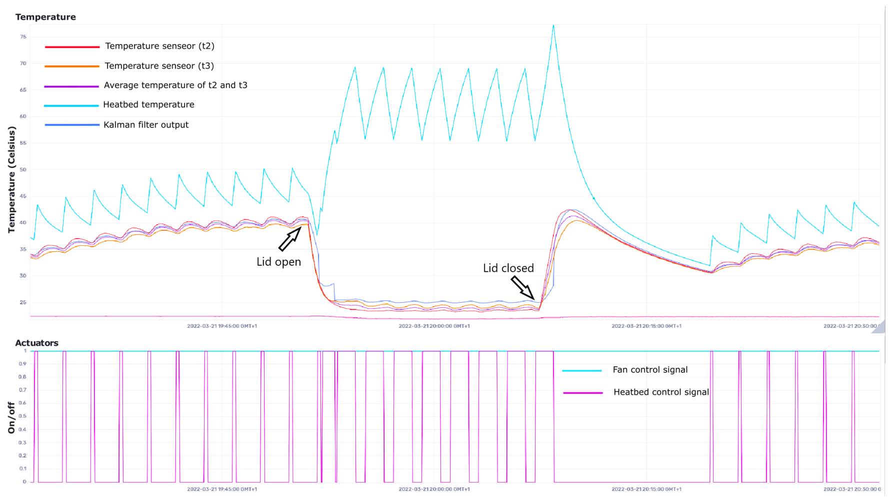


**Fig. 18** Self-adaptation Experiments (adapted from [1]).

## 4.8 Other DT Cases

In this Section different DT services were considered in the context of the incubator case-study, which is arguably a rather small system. DT showever, could be adopted in all kinds of domains ranging from manufacturing to mobile robotics. In the following, examples from these domains are described briefly.

### The Desktop Robotti

The Desktop Robotti is a small prototype of an agricultural robot, namely the Robotti, produced by AgroIntelli [45] (both shown in Figure 19). This robot makes it possible to carry out testing activities in the lab, with low cost and low risk of damage. At the time of writing a DT is being built for the

Desktop Robotti in order to provide services such as monitoring and reconfiguration when undesired scenarios unfold. In more detail, the DT would follow the motion of the robot, i.e., the PT, by comparing simulated values of the robot's position to the live feed coming from the robot itself, also referred to as parallel operation. Furthermore, the DT should handle collision zones that may arise when two or more robots drive too close to one another.

The DT and the Desktop Robotti communicate through the RabbitMQ server, with the information flowing in both directions. The Desktop Robotti sends its state to the DT continuously, which consists of its pose and velocity. The DT sends control commands to the robot when necessary, e.g. emergency stops if the robot is close to collision. The robot can also be directly operated by a user, by-passing the DT connection.

The main sensor of the robot is a 2D lidar, RPlidar A1, which provides planar scans of the surroundings and can be used in conjunction with the navigation stack in the Robot Operating System (ROS) [46]. This information, alongside the robot's position is vizualized using the RVIZ tool as seen in Figure 20.

The models in the DT represent the motion of the robot, and consist in a kinematics and actuation model. The basis for the kinematics is a bicycle model, where it is assumed that there are virtual wheels in the centres of the front and rear axes. The model is expressed with equations, where the current state (pose) of the robot is computed from a past value. The inputs are the velocity and the steering angle, and the output is the change in the pose. The reader is referred to [47] for detailed derivations.

The actuation model on the other hand relates to the DC motors, and is a first order system. Note that, since the motors are controlled by input voltage, and produce a mechanical output, there is not a straightforward correlation to speed. As such calibration needs to be carried out.

The basic DT service for the Desktop Robotti is monitoring of the robot's pose and its comparison to the simulated pose in the DT [48]. A simple example consisting in the robot moving through a three point turn is shown in Figure 21.

Note that results shown in Figure 21 correspond to a 30 seconds run. It is possible to observe a discrepancy of the signals already near the second turn, deepening by the third. The root cause of this is that the change in the steering angle is considered instantaneous in the DT. The need for high fidelity models is highlighted in this case.

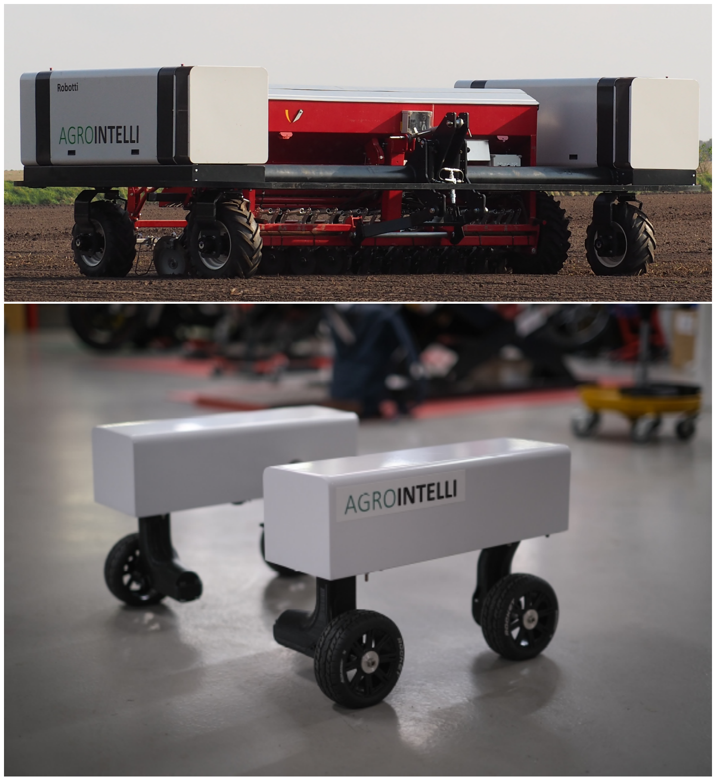


**Fig. 19** The Robotti field robot (top) and its small replica (bottom) [45].

The DT for the Desktop Robotti can also be used for run-time reconfiguration of the behaviour of the robot [49]. One way in which re-configuration can be supported is by updating models in the DT during run-time, and is usually referred to as model swapping. When developers build new models, either to be enclosed in the DT or eventually deployed to run natively on the PT, they can be safely tested initially in the DT. These new models can be added in a warm-up mode in parallel to the corresponding old ones, where their outputs do not affect change in the DT/PT system, until some safety thresholds are satisfied. When it is deemed safe to swap in the new models, their outputs are activated and the old models are removed. .

Assume a scenario, as depicted in Figure 22 (left), where the DT is composed of the controller of the model and a data broker which allows bi-directional communication with the PT [49]. The user is also able to send control commands via the separate interface, specifically velocity commands. Assume further that a new controller has been devised able to cap the velocity commands send by the user, should they exceed a predetermined safety value. The new controller is swapped in during run-time, only after ensuring that the robot's velocity is under a certain value, as can be observed in Figure 23.

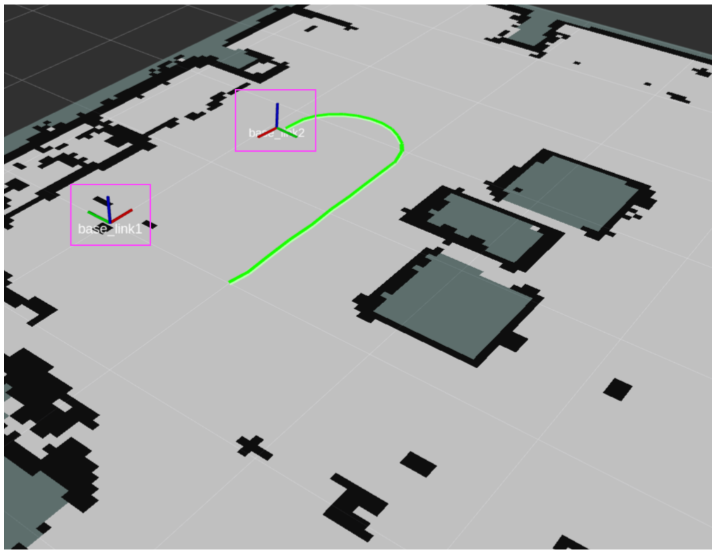


**Fig. 20** Two robots (within the magenta boxes) visualized in RVIZ (taken from [1]). The green line denotes the path traversed by the robot.

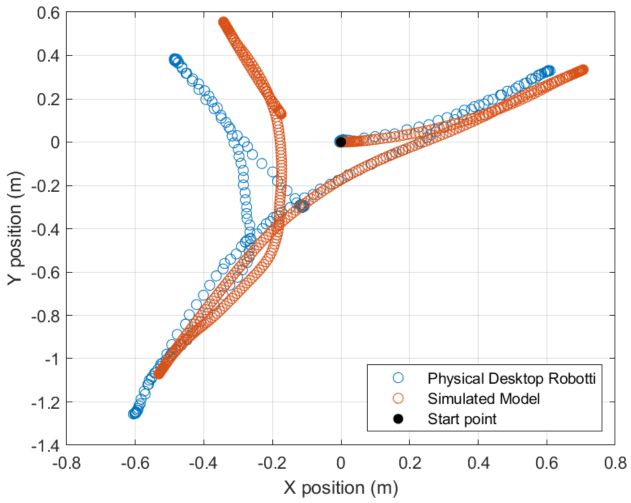


**Fig. 21** Real and simulated pose of the PT during a three point turn. Taken from [48].

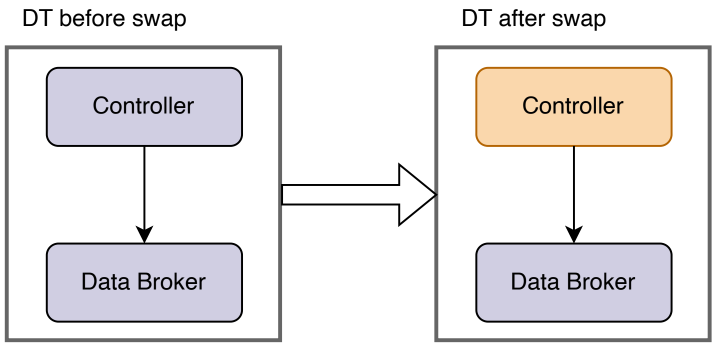


**Fig. 22** The constellation of the DT before (left) and after (right) the swap (adapted from [1]).

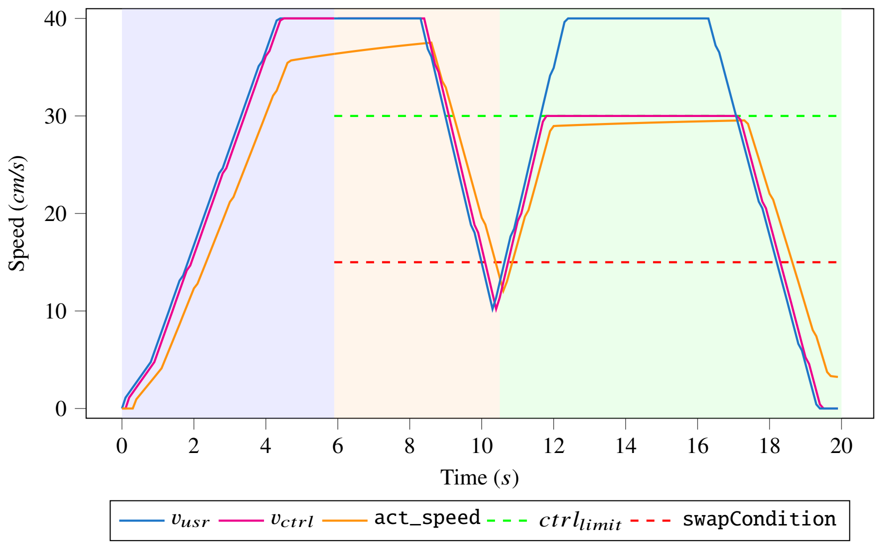


**Fig. 23** Speed adaptation through replacing at runtime the controller in the DT (adapted from [49]). Blue solid line denotes the velocity specified by the user ($v_{usr}$), the red solid line indicates is the velocity set by the controller ($v_{ctrl}$), and the solid orange line denotes the actual speed of the CPS (`act_speed`). Blue background from time 0 to 5.9 utilises the controller without a speed cap. The new specification is introduced and evaluated at time 5.9 (orange background), defining the model swap. The new controller is swapped in at time 10.4 when the speed is below a pre-defined safety threshold, in red (i.e., lower threshold to perform the model swap $swapCondition$), and the new speed cap $ctrl_{limit}$ enforced by the new controller in green.

Note that in the above description we do not delve in the technical implementation details, but rather aim to provide a general overview of potential DT services. The reader is pointed to [48, 49] for a details on how these services are implemented.

## The Flex-cell

The Flex-cell consists of two robotic arms from different manufacturers, which are intended to perform a cooperative assembly task (Figure 24). There are four main components in the PT: Kuka LBR iiwa 7 and UR5e robotic arms, as well as the OnRobot RG6 and OnRobot 2FG7 grippers. This setup enables researchers to explore among others: composable DTs, cooperative systems with synchronous motions, multiple abstraction levels of attributes and operations, systems with complex kinematics and dynamics, multiple applications such as assembly task, assembly line optimisation etc.

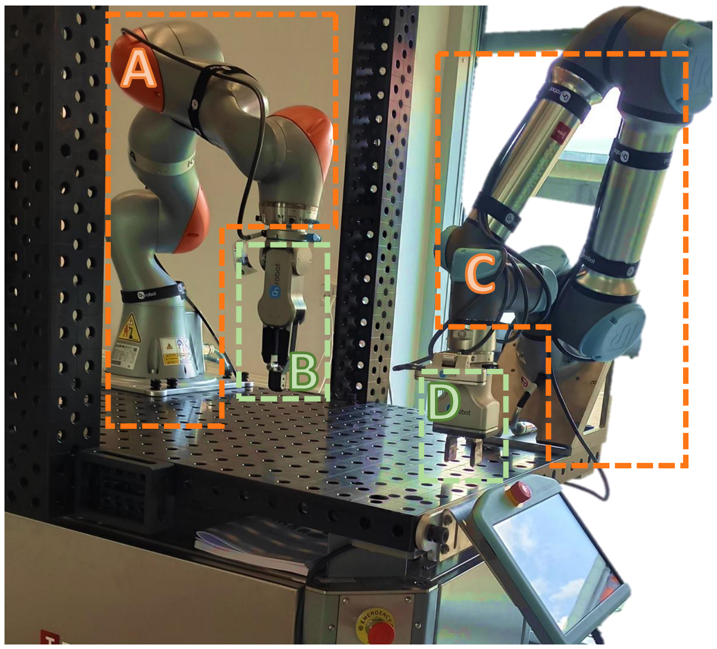


**Fig. 24** The Flex-cell, Kuka LBR iiwa 7 (A), UR5e (C), OnRobot RG6 (B), and the OnRobot 2FG7 (D). Adapted from [1].

The DT for the Flex-cell is a composition of sub-DTs [50], each for robotic arm and gripper, which is expressed through a isComposedOf relationship. In addition each sub-DT is described by attributes, operations, behaviours and relationships. Simulations are available only for the robotics arms, whereas for the grippers there is only static and structural information available. Visual representations for the simulation are provided through Unity game engine[2] (Figure 25). The Flex-cell DT is a composition of seven DT instances, one for each robot and gripper, two intermediate compositions each consisting of a robotic arm and their gripper, and a final composition of the two previous compositions.

Commands can be sent to the robotic arms and grippers, further processed in the internal controllers of each. In regards to the robotic arms, different motion types could be performed, e.g., cartesia, joint, linear, circular, as well as other parameter updates, e.g., speed and acceleration. These actions can also be performed manually using tech pendants. Commands to the grippers are handled in a similar way, where the operator can update default values for the gripping force or opening of the gripper.

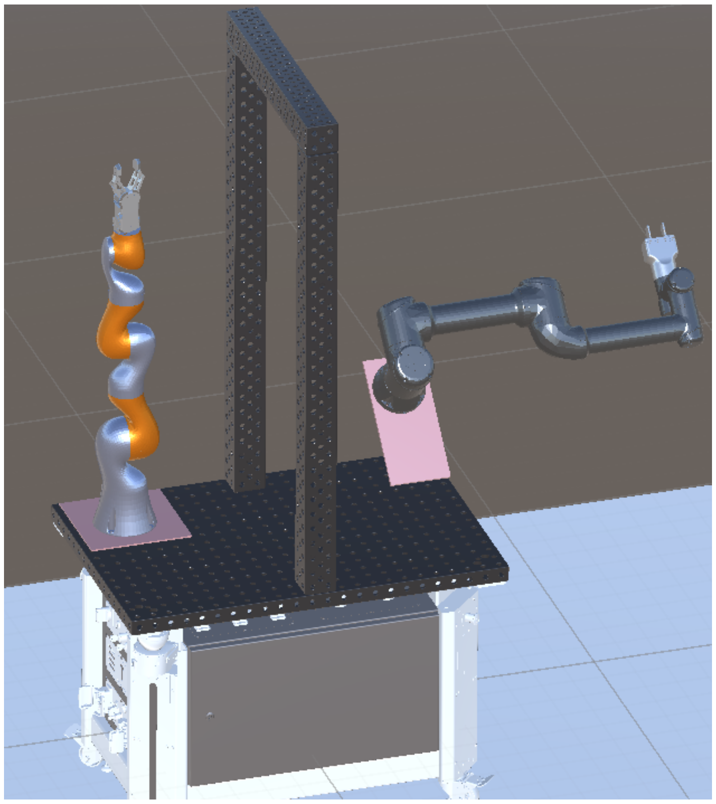


**Fig. 25** The Flex-cell visualization in Unity. Taken from [1].

Several sensors are available, making it possible to collect data regarding the current and target positions, torque, current, voltage, speed, acceleration etc. Proximity

2 https://unity.com/

sensors are also in place, which trigger safety stops if humans are too close or touch robot inadvertently or not.

A DT Manager [51] handles the communication between the DT and PT. Moving commands can also be performed, where the PT is provided with joint angles by the DT. The DT for the Flex-cell offers a range of services as follows:

1. Cooperative execution, where motion commands can be executed synchronously on both robots and corresponding simulations.
2. Trajectory visualization through Unity, aiding the operators working with the Flex-cell understand what the robots are doing.
3. Discrete space-related commands, which are used to translate the target position (X,Y,Z) to joint angles for the robots.
4. Deviation checking, as well as collision detection can also be inspected by an operator during the execution of a task.
5. Virtual commisioning, where what-if analysis is performed, with the resulting strategies validated on the virtual Flex-cell before being deployed to the PT.

An example for the cooperative run is shown in Figure 26, where the real-robots are run in parallel to their simulations. The joint positions for each are shown over time, as the robots move between two discrete waypoints. Note that the simulation starts with 0 values, whereas the PT starts from the values from its previous run.

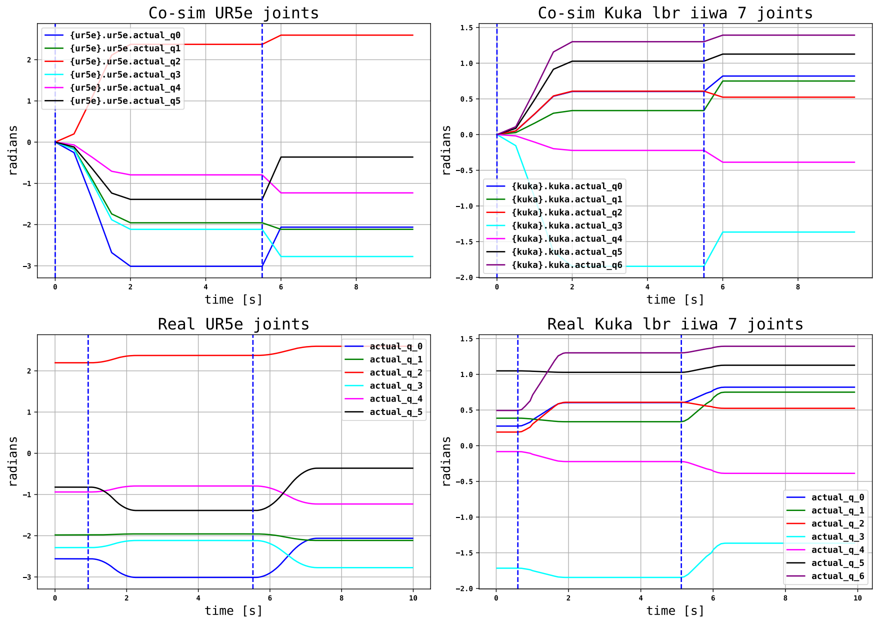


**Fig. 26** Joint angles from the real robots and respective simulations for cooperative execution. Movement commands are indicated with vertical dashed lines (adapted from [1]).

As with the Desktop Robotti, the models used for simulation need to have medium to high fidelity for the services provided from the DT to be of value. Failure to achieve an adequate fidelity level can, not only lead to poor performance, but also to dangerous situations where robots collide with each other and cause damage to objects, or the Flex-cell itself. A few limitations are already present in the current models as the kinematics do not include the grippers. In addition trajectory generation relies on assumption on motion speed, without considering system dynamics, leading to potentially high errors.

## 5 Future Directions for Digital Twins

DTs have been around for decades, and are demonstrating in lrecentater years a growing interest from industry and academia. This is due to the promise of DT-enabled systems being easier to analyse, monitor, and re-configure PTs at runtime, thus reducing their operational downtime. New technologies are emerging every day, allowing to harness larger amounts of data in real-time, bringing the promise of combination of models, data, and computation, as well as the respective benefits even closer. Nevertheless, several fundamental gaps need to be overcome. To do so, we consider the following directions for future DT research:

**Foundations:** In order for DTs to be useful, their predictions need to be trusted, such that they can be used to guide decision-making. To this end, it is necessary to understand the limits of the predictions, and master uncertainties.
However, due to the hich dimensionality of DTs – from data capture at the PT, to data communication and finally modelling and predictions realised by the different DT services – the accuracy of the DT is limited, hindering to overcome uncertainties between PTs and DTs. A few key qualities concerning models have been identified by preliminary research [52], including relevance, verifiability, substitutability, and faithfulness. These are expected to impact DT services, given that the latter would rely on different kinds of models. The impact relationship however is not clear at the time of writing. Understanding the limits of predictions, factually means being able to answer the following question: "What is the prediction accuracy of a DT with a model with complexity order **R** and a set of **P** properties testing **A** abnormality criteria using information from **S** sensors sharing a communication channel of capacity **C** and a computational resource of **F** flops?" [1].
The accuracy of predictions is inherently tied to uncertainties. Depending on the domain, different techniques have been used traditionally to tackle uncertainty, in terms of its representation and propagation models, such as reachability analysis [53], Monte-Carlo simulations [54], stochastic differential equations [55] , and sensitivity analysis [56]. With respect to DTs however, representing uncertainty for the different models, their propagation through the different layers, as well as

the impact on the operation of such layers, remains a challenge. Further research is required to model the effects of different uncertainty sources.

**Platform:** Developing and maintaining DTs is not a trivial matter, considering also situations where DTs are being developed in parallel to their PTs. The engineering of such systems should become affordable and sustainable.

Drawing from the experience of Model-Driven Engineering (MDE) could be quite beneficial dealing with challenges in DT engineering, such as the management of heterogeneous components (models), synchronisation between DT and PT, and support for multi-stakeholder and multi-discipline engineering [57]. Any model-based framework should consist in a basic architecture, language concepts and services that enable the integration of different components in a DT, while also making it possible to couple a PT to it. Standards like the Functional Mockup Interface (FMI), for specifying interfaces for models built with different tools, could have potential for DT engineering [58]. More research should also go towards languages for defining DTs, that support portability, reuse, as well as validation and verification [59], where a starting point could be the DT definition language (DTDL), for Azure DTs[3].

Techniques that integrate ontologies and knowledge graphs will also be needed, that will serve as a semantic backbone for the DT, in terms of the alignment, integration, and description of its components. In addition, tools and techniques to support distributed simulations and execution of DT components on a distributed hardware infrastructure will be necessary for heavy computations such as what-if simulations. Platforms should also support the management of the full life cycle of DTs and its assets, as well as enabling collaborative development of DT-enabled systems.

**Autonomy:** While PTs can themselves display certain levels of autonomy, DTs could also be viewed as extending their PTs autonomy, e.g., by carrying out re-configuration actions when a fault is encountered. The autonomous behaviour needs to be trustworthy before we can deploy it in the real-world. The difficulty lies in how autonomy of different components, consider not only a PT and its DT, but networks of such systems, namely multi-agent systems, is affected by planned and emerging interactions. In addition, the role of the human needs to be specified. Human involvement could be regulated based on the fidelity of the DT's predictions at a given time. Should the trust in the predictions not be sufficiently high as compared to the consequences of a bad predictions, the human could adopt a leading role, and vice-versa. This would further require that a human has good situational awareness before making any decisions. Autonomy is also coupled with intelligence, and capabilities like reasoning, learning, and self-adaptation, leading to what is called a cognitive DT, able to handle unpredicted complex situations and apply mitigation strategies as required [60]. To realise this vision further research is needed in perception and prediction, as well as uncertainties, connected to the ability of the DT to evaluate its own simulation versus reality gap.

[3] https://azure.github.io/opendigitaltwins-dtdl/

**Composition:** Platforms that support the composition of DTs are rather useful, both pertaining near-identical to heterogeneous systems. Naturally, the question how such composition should be organised is raised, considering that organisations involved in such efforts are not necessarily only collaborators but also competitors.
As the fidelity of the DT increases, its constituent models will reflect quite closely the intellectual property (IP) of its organisation. Thus it is crucial to consider how to secure IPs, otherwise competing organisations will not be meaningful collaborators in developing composable DTs. More research is needed on standardised interfaces as previously mentioned, as well as on the collaborative aspect of DTs. Experience from the multi-agent system domain promises to be quite useful for supporting coordination as well as emergent interactions between systems (DTs and/or PTs).

## 6 Concluding Remarks

This chapter has illustrated how DT-enabled systems can be used in an industrial setting. A definition of DTs has been provided and it has been explained how such DTs can be constructed. It is important to understand which services that provides value for a specific CPS and to what extent it gives most value for the users and/or the manufacturers. There is clearly a trend globally to change to a more service-oriented offering, where customers pay for the time they use a product, rather than buying the product. Very different types of services for DTs has been presented. Naturally this is an evolving world so there are also numerous future directions for DTs. Some of these have also been touched upon in this chapter.

**Acknowledgements** If you want to include acknowledgments of assistance and the like at the end of an individual chapter please use the `acknowledgement` environment – it will automatically be rendered in line with the preferred layout.

## References

1. Peter Gorm Larsen, John Fitzgerald, and Cláudio Gomes. *Engineering Digital Twins for Cyber-Physical Systems*, pages 3–17. Springer International Publishing, Cham, 2024.
2. Michael Grieves and John Vickers. Digital Twin: Mitigating Unpredictable, Undesirable Emergent Behavior in Complex Systems. In Franz-Josef Kahlen, Shannon Flumerfelt, and Anabela Alves, editors, *Transdisciplinary Perspectives on Complex Systems*, pages 85–113. Springer International Publishing Switzerland, August 2017.
3. P. Coveney and R. Highfield. *Virtual You: How Building Your Digital Twin Will Revolutionize Medicine and Change Your Life*. Princeton University Press, 2023.
4. Asaf Tzachor, Soheil Sabri, Catherine E. Richards, Abbas Rajabifard, and Michele Acuto. Potential and limitations of digital twins to achieve the Sustainable Development Goals. *Nature Sustainability*, 5:822–829, October 2022.

5. Werner Kritzinger, Matthias Karner, Georg Traar, Jan Henjes, and Wilfried Sihn. Digital twin in manufacturing: A categorical literature review and classification. *IFAC-PapersOnLine*, 51:1016–1022, 01 2018.
6. Dimitrios Piromalis and Antreas Kantaros. Digital twins in the automotive industry: The road toward physical-digital convergence. *Applied System Innovation*, 5, 07 2022.
7. Ahmad K. Sleiti, Jayanta S. Kapat, and Ladislav Vesely. Digital twin in energy industry: Proposed robust digital twin for power plant and other complex capital-intensive large engineering systems. *Energy Reports*, 8:3704–3726, 2022.
8. F. Naseri, S. Gil, C. Barbu, E. Cetkin, G. Yarimca, A.C. Jensen, P.G. Larsen, and C. Gomes. Digital twin of electric vehicle battery systems: Comprehensive review of the use cases, requirements, and platforms. *Renewable and Sustainable Energy Reviews*, 179:113280, 2023.
9. Christos Pylianidis, Sjoukje Osinga, and Ioannis N. Athanasiadis. Introducing digital twins to agriculture. *Computers and Electronics in Agriculture*, 184:105942, 2021.
10. John Fitzgerald, Cláudio Gomes, and Peter Gorm Larsen, editors. *The Engineering of Digital Twins*. Springer, 2024.
11. George Em Karniadakis, Ioannis G Kevrekidis, Lu Lu, Paris Perdikaris, Sifan Wang, and Liu Yang. Physics-informed machine learning. *Nature Reviews Physics*, 3(6):422–440, 2021.
12. Warren S McCulloch and Walter Pitts. A logical calculus of the ideas imminent in nervous activity. bulletin of mathematical biophysics, vol. 5, pp. 115-133. *Journal of Symbolic Logic*, 9, 1943.
13. David Harel. Statecharts: A visual formalism for complex systems. 8(3):231–274.
14. D. Bjørner and C.B. Jones, editors. *The Vienna Development Method: The Meta-Language*, volume 61 of *Lecture Notes in Computer Science*. Springer-Verlag, 1978.
15. D. Bjørner and C.B. Jones, editors. *Formal Specification and Software Development*. Prentice-Hall International, 1982.
16. John Fitzgerald and Peter Gorm Larsen. *Modelling Systems – Practical Tools and Techniques in Software Development*. Cambridge University Press, The Edinburgh Building, Cambridge CB2 2RU, UK, 1998. ISBN 0-521-62348-0.
17. Cláudio Gomes, Casper Thule, David Broman, Peter Gorm Larsen, and Hans Vangheluwe. Co-simulation: a Survey. *ACM Comput. Surv.*, 51(3):49:1–49:33, May 2018.
18. Torsten Blockwitz, Martin Otter, Johan Åkesson, Martin Arnold, Christoph Clauss, Hilding Elmqvist, Markus Friedrich, Andreas Junghanns, Jakob Mauss, Dietmar Neumerkel, Hans Olsson, and Antoine Viel. Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *Proc. 9th International Modelica Conference*. Linköping University Electronic Press, 2012.
19. T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. Clauss, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel. The Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Models. In *Proceedings of the 9th International Modelica Conference*, Munich, Germany, September 2012.
20. Andreas Junghanns, Torsten Blochwitz, Christian Bertsch, Torsten Sommer, Karl Wernersson, Andreas Pillekeit, Irina Zacharias, Matthias Blesken, Pierre Mai, Klaus Schuch, Christian Schulze, Cláudio Gomes, and Masoud Najafi. The Functional Mock-up Interface 3.0 - New Features Enabling New Applications. In *Proceedings of the 14th International Modelica Conference*, online, 2021. Linköping University Electronic Press, Linköpings Universitet.
21. Mirgita Frasheri, Henrik Ejersbo, Casper Thule, Cláudio Gomes, Jakob Levisen Kvistgaard, Peter Gorm Larsen, and Lukas Esterle. Addressing time discrepancy between digital and physical twins. *Robotics and Autonomous Systems*, 161:104347, 2023.
22. Santiago Gil, Peter H. Mikkelsen, Cláudio Gomes, and Peter G. Larsen. Survey on open-source digital twin frameworks–A case study approach. *Software: Practice and Experience*, 54(6):929–960, 2024.
23. Hao Feng, Cláudio Gomes, Casper Thule, Kenneth Lausdahl, Michael Sandberg, and Peter Gorm Larsen. The Incubator Case Study for Digital Twin Engineering. February 2021.
24. Aaron John Buhagiar, Leo Freitas, William E. Scott III, and Peter Gorm Larsen. Digital twins for organ preservation devices. In Tiziana Margaria and Bernhard Steffen, editors, *ISoLA 2020*, volume 13704 of *Lecture Notes in Computer Science*, pages 22–36. Springer, 2022.

25. K. H. Steinkraus, Y. B. Hwa, J. P. Van Buren, M. I. Provvidenti, and D. B. Hand. Studies on tempeh. An Indonesian fermented soybean food. 25:777–788.
26. Alexie Dingli and Foaad Haddod. Interacting with Intelligent Digital Twins. In Aaron Marcus and Wentao Wang, editors, *Design, User Experience, and Usability. User Experience in Advanced Technological Environments*, Lecture Notes in Computer Science, pages 3–15, Cham, 2019. Springer International Publishing.
27. Andreas Künz, Sabrina Rosmann, Enrica Loria, and Johanna Pirker. The potential of augmented reality for digital twins: A literature review. In *2022 IEEE conference on virtual reality and 3D user interfaces (VR)*, pages 389–398. IEEE, 2022.
28. Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications. In Ezio Bartocci and Yliès Falcone, editors, *Lectures on Runtime Verification*, volume 10457 of *Lecture Notes in Computer Science*, pages 135–175. Springer International Publishing, Cham, 2018.
29. Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems*. Springer-Verlag, 1992.
30. Yliès Falcone, Srđan Krstić, Giles Reger, and Dmitriy Traytel. A taxonomy for classifying runtime verification tools. *International Journal on Software Tools for Technology Transfer*, 23(2):255–284, April 2021.
31. Andreas Bauer, Martin Leucker, and Christian Schallhart. The good, the bad, and the ugly, but how ugly is ugly? In Oleg Sokolsky and Serdar Taşıran, editors, *Runtime Verification*, pages 126–138, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
32. Fatemeh Kakavandi, Roger De Reus, Negar Heidari, Alexandros Iosifidis, and Peter Gorm Larsen. Product quality control in assembly machine under data restricted settings. In *IEEE International Conference on Industrial Informatics*, pages 735–741, 2022.
33. Benny Hockner, Petra Hofstedt, Sascha Kaltschmidt, Peter Sauer, and Thilo Vörtler. Design space exploration for cyber physical system design using constraint solving. In *Proceedings of the 2013 Forum on specification and Design Languages, FDL 2013, Paris, France, September 24–26, 2013*, pages 1–4. IEEE, 2013.
34. Kalyanmoy Deb. Multi-objective optimisation using evolutionary algorithms: an introduction. In *Multi-objective evolutionary optimisation for product design and manufacturing*, pages 3–34. Springer, 2011.
35. Nadir K Salih, D Satyanarayana, Abdullah Said Alkalbani, and R Gopal. A survey on software/hardware fault injection tools and techniques. In *2022 IEEE Symposium on Industrial Electronics & Applications (ISIEA)*, pages 1–7. IEEE, 2022.
36. Jean Arlat. *Validation de la sûreté de fonctionnement par injection de fautes: méthode, mise en oeuvre, application*. PhD thesis, Toulouse, INPT, 1990.
37. Yangyang Yu, Bertrand Bastien, and Barry W Johnson. A state of research review on fault injection techniques and a case study. In *Annual Reliability and Maintainability Symposium, 2005. Proceedings.*, pages 386–392. IEEE, 2005.
38. Rolf Isermann. Supervision, fault-detection and fault-diagnosis methods—an introduction. *Control engineering practice*, 5(5):639–652, 1997.
39. Sujit Rokka Chhetri, Sina Faezi, Arquimedes Canedo, and Mohammad Abdullah Al Faruque. QUILT: Quality inference from living digital twins in IoT-enabled manufacturing systems. In *Proceedings of the International Conference on Internet of Things Design and Implementation*, IoTDI '19, pages 237–248. Association for Computing Machinery, 2019.
40. Jinjiang Wang, Lunkuan Ye, Robert X Gao, Chen Li, and Laibin Zhang. Digital twin for rotating machinery fault diagnosis in smart manufacturing. *International Journal of Production Research*, 57(12):3920–3934, 2019.
41. Kamal Medjaher, Noureddine Zerhouni, and Rafael Gouriveau. *From prognostics and health systems management to predictive maintenance 1: Monitoring and prognostics*. John Wiley & Sons, 2016.
42. Nam-Ho Kim, Dawn An, and Joo-Ho Choi. Prognostics and health management of engineering systems. *Switzerland: Springer International Publishing*, 2017.

43. Jeffrey O Kephart and David M Chess. The vision of autonomic computing. *Computer*, 36(1):41–50, 2003.
44. Hao Feng, Cláudio Gomes, Santiago Gil, Peter Høgh Mikkelsen, Daniella Tola, Michael Sandberg, and Peter Gorm Larsen. Integration of the MAPE-K Loop in Digital Twins. IEEE, 2022.
45. Frederik Foldager, Ole Balling, Carl Gamble, Peter Gorm Larsen, Martin Boel, and Ole Green. Design Space Exploration in the Development of Agricultural Robots. In *AgEng conference*, Wageningen, The Netherlands, July 2018.
46. Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y Ng. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, 3, page 5, 2009.
47. Jacob Odgaard Hausted, Jakob Levisen Kvistgaard, and Gill Lumer-Klabbers. Towards a Digital Twin for an Autonomous Robot. Aarhus University, BSc thesis, December 2020.
48. Gill Lumer-Klabbers, Jacob Odgaard Hausted, Jakob Levisen Kvistgaard, Hugo Daniel Macedo, Mirgita Frasheri, and Peter Gorm Larsen. Towards a digital twin framework for autonomous robots. In *SESS: The 5th IEEE International Workshop on Software Engineering for Smart Systems*. COMPSAC 2021, IEEE, July 2021.
49. Henrik Ejersbo, Kenneth Lausdahl, Mirgita Frasheri, and Lukas Esterle. Dynamic Runtime Integration of New Models in Digital Twins. In *2023 IEEE/ACM 18th Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS)*, pages 44–55, 2023.
50. Santiago Gil, Peter H Mikkelsen, Daniella Tola, Casper Schou, and Peter G Larsen. A Modeling Approach for Composed Digital Twins in Cooperative Systems. In *2023 IEEE 28th International Conference on Emerging Technologies and Factory Automation (ETFA)*, pages 1–8. IEEE, 2023.
51. Daniel Lehner, Santiago Gil, Peter H. Mikkelsen, Peter G. Larsen, and Manuel Wimmer. An architectural extension for digital twin platforms to leverage behavioral models. In *2023 IEEE 19th International Conference on Automation Science and Engineering (CASE)*, pages 1–8, 2023.
52. Bentley Oakes, Cláudio Gomes, Peter Gorm Larsen, Joachim Denil, Julien DeAntoni, João Cambeiro, and John Fitzgerald. Examining Model Qualities and Their Impact on Digital Twins. In *Annual Modelling and Simulation Conference*, pages 220–232.
53. Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul Jackson, editors, *Computer Aided Verification*, pages 167–170, Berlin, Heidelberg, 2010. Springer Berlin Heidelberg.
54. Jim Woodcock, Cláudio Gomes, Hugo Daniel Macedo, and Peter Gorm Larsen. Uncertainty quantification and runtime monitoring using environment-aware digital twins. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends*, pages 72–87, Cham, 2021. Springer International Publishing.
55. Tugcem Partal Mustafa Bayram and Gulsen Orucova Buyukoz. Numerical methods for simulation of stochastic differential equations. *Advances in Difference Equations*, (17), January 2018.
56. Yi Chou and Sriram Sankaranarayanan. Bayesian parameter estimation for nonlinear dynamics using sensitivity analysis. In *Proceedings of the Twenty-Eighth International Joint Conference on Artificial Intelligence, IJCAI-19*, pages 5708–5714. International Joint Conferences on Artificial Intelligence Organization, 7 2019.
57. Francis Bordeleau, Benoit Combemale, Romina Eramo, Mark van den Brand, and Manuel Wimmer. Towards model-driven digital twin engineering: Current opportunities and future challenges. In Önder Babur, Joachim Denil, and Birgit Vogel-Heuser, editors, *Systems Modelling and Management*, pages 43–54, Cham, 2020. Springer International Publishing.
58. Hao Feng, Cláudio Gomes, Michael Sandberg, Hugo Daniel Macedo, and Peter Gorm Larsen. Under what conditions does a digital shadow track a periodic linear physical system? In Antonio Cerone, Marco Autili, Alessio Bucaioni, Cláudio Gomes, Pierluigi Graziani, Maurizio Palmieri, Marco Temperini, and Gentiane Venture, editors, *Software Engineering and*

*Formal Methods. SEFM 2021 Collocated Workshops*, pages 143–155, Cham, 2022. Springer International Publishing.

59. James Moyne, Yassine Qamsane, Efe C. Balta, Ilya Kovalenko, John Faris, Kira Barton, and Dawn M. Tilbury. A requirements driven digital twin framework: Specification and opportunities. *IEEE Access*, 8:107781–107801, 2020.
60. Lukas Esterle, Cláudio Gomes, Mirgita Frasheri, Henrik Ejersbo, Sven Tomforde, and Peter G. Larsen. Digital twins for collaboration and self-integration. In *2021 IEEE International Conference on Autonomic Computing and Self-Organizing Systems Companion (ACSOS-C)*, pages 172–177. IEEE.