Digital Twins: a Briefing for Formalists

John S. Fitzgerald[1][0000−0001−7041−1807], Cláudio Gomes[2][0000−0003−2692−9742], Peter Gorm Larsen[2][0000−0002−4589−1500], Mikkel Schmidt Andersen[2][0009−0006−8678−1726], Santiago Gil[2][0000−0002−1789−531X], and Morten Haahr Kristensen[2][0009−0008−8467−7567]

[1] School of Computing, Newcastle University, UK
John.Fitzgerald@newcastle.ac.uk
[2] Dept. of Electrical and Computer Engineering, Aarhus University, DK
{claudio.gomes,pgl,msa,sgil,mhk}@ece.au.dk

Abstract. Digital twins couple computational models with physical systems to enable services such as predictive reasoning and runtime analysis that support decision-making for maintenance, optimisation and system evolution. They create a technically rich setting for applying formal methods throughout the system lifecycle beyond design-time. This tutorial introduces the elements of digital twin technology, identifying emerging opportunities for formal techniques, including the verification of evolving models under uncertainty, integration of heterogeneous models and rigorous composition of digital twin-enabled systems.

Keywords: Digital Twins · Systems Engineering · Formal Methods

1 Introduction

The *Digital Twin (DT)* concept is part of a global megatrend of digital transformation [35] that has attracted attention and investment across many domains. At its heart is the use of digital models — often multiple heterogeneous models — beyond design time to provide insight into the current state and anticipated future of a real-world system. The DT adds value by offering services such as visualisation, monitoring and fault diagnosis, optimisation and predictive maintenance, informing human or automated decision-making that in turn may change the physical system. Realising this vision depends on maintaining alignment between the digital and physical twins via a sensing and communication infrastructure.

The engineering of dependable DTs presents significant challenges, including the management of heterogeneous models, reasoning under uncertainty arising from real-world interactions, rigorous validation and verification of DTs, and dependable composition of DT-enabled systems. Formal Methods (FMs) are well positioned to address these issues.

This tutorial introduces DTs with an emphasis on opportunities for future FM research and innovation, informed by the authors' work in this emerging area. It builds on prior publications, including a book and tutorial on DT Engineering [24, 33], but adopting FM perspective and extending prior tutorial

publications with insights from more recent case studies and roadmaps such as that of the US National Academies of Science, Engineering and Medicine [51].

We outline core DT concepts from the perspective of model-based systems engineering (Section 2), give an overview of achievements and challenges in the engineering of DTs (Section 3), consider examples formal techniques in DT Engineering to date (Section 4) and identify promising directions for future FM contributions (Section 5). Section 6 provides a few concluding remarks.

2 Fundamentals

2.1 DT-enabled Systems

In this tutorial, we adopt a systems perspective. A system is an "arrangement of parts or elements that together exhibit a stated behaviour or meaning that the individual constituents do not" [36]. A *system boundary* separates the system from its *environment*. The system's interactions with that environment determine its observable *behaviour*. Our interest is in DTs of *Cyber-Physical Systems (CPSs)*, whose elements combine computational and physical processes.

A DT is a digital system that replicates a real-world entity termed the *Physical Twin (PT)*[3]. The DT and PT are linked by a *communications infrastructure* through which data flows in both directions, allowing changes in one to update the other. This view aligns with emerging standards [37–39], while explicitly permitting the DT to influence the PT, potentially autonomously. A DT that only mirrors the PT without affecting it is often termed a *Digital Shadow*. We refer to the composition of PT, communications infrastructure and DT as the *DT-enabled system (DTeS)* (Fig. 1).

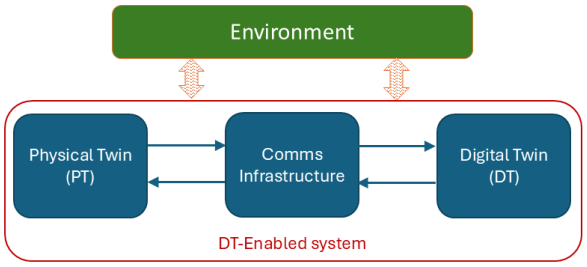


Fig. 1. Simplified overview of a DTeS. External stakeholders omitted.

A DT uses models of the PT alongside incoming data to offer its stakeholders services such as visualisation, monitoring, fault diagnosis, predictive maintenance, what-if analysis and reconfiguration. These services form the building

[3] Despite the name, the *Physical* Twin is often itself a CPS.

blocks of the DT discussed in Section 3 below. Note that the DT is different from a model-based controller in that it adds value to that of the PT alone; the PT should operate satisfactorily even if the DT is absent.

2.2 A Basic Example: the Tempeh Incubator

We introduce a small example to illustrate concepts introduced so far. For full details we refer the reader to a previous tutorial [33] and book [24]. Our example is based on an incubator for producing the fermented food tempeh. In tempeh production, cooked soybeans are inoculated with a fungus and fermented at a controlled temperature for 24-48 hrs. The fermentation initially absorbs heat, but later the internal temperature rises as the fungus grows rapidly. Just after the temperature passes a peak, the tempeh is removed for consumption.

The (cyber-)physical incubator (Figure 2) consists of a styrofoam box with a heatbed, fan and temperature sensors, controlled by a Raspberry Pi. An online video shows the real-world setup[4]. The incubator's control system, implemented in Python, includes a communication server, a low-level driver and a controller that manages the heatbed and fan, with a basic on-off control strategy. A communication server facilitates message exchange between the controller and a low-level driver. The driver manages communications with sensors and actuators, reading current temperature data and checking for commands from the controller via the communication server. The controller is a form of bang-bang controller which keeps temperature between lower and upper limits. A user of

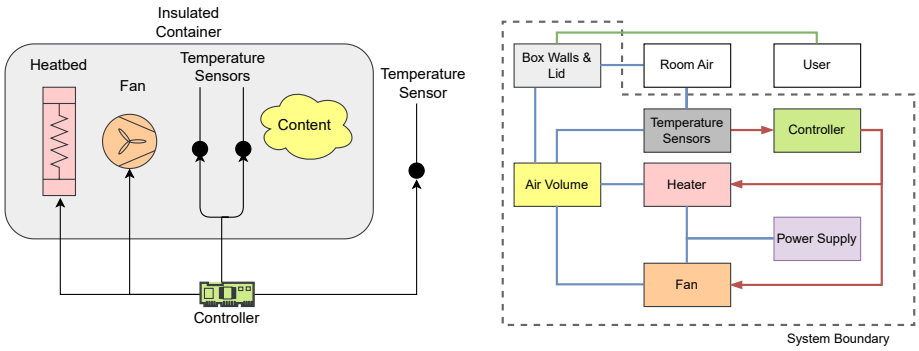


Fig. 2. Informal schematic of tempeh incubator showing cyber-physical setup (left) and abstraction of the CPS's elements and their interactions. From [24].

this setup can ferment tempeh satisfactorily, but we envisage a DT that extends the incubator's capabilities, interfacing with stakeholders through a connected device such as a smartphone, to offer improved control, higher consistency and

[4] `https://youtu.be/gG4za7iPY0I`

better quality of the final product. The services offered by such a DT may include:

- visualisation of the measured state of the fermentation (e.g., via a dashboard or a 4D graphical view);
- monitoring in order to detect anomalies such as the opening of the incubator lid;
- decision support, for example in selecting from among alternative strategies for replacing incubator components, or trading off parameters (e.g., temperature error and actuator working life);
- reconfiguration, such as switching in alternative control strategies when the incubator is used outside the current assumptions.

Creating a DTeS that can deliver such services requires an understanding of the ecosystem of models that go to make up the DT, each with their own purposes, assumptions, formalisms and levels of abstraction. It must also take into account the connecting infrastructure an other sources of uncertainty and the potential for dynamic adaptation. In all of these areas, there are opportunities for contributions from FMs.

3 The Building Blocks of a DT

The emerging field of *DT Engineering* encompasses the analysis, specification, design, implementation, deployment, maintenance and disposal of DTeSs. It is inherently interdisciplinary in that DTeSs may be developed by teams of specialists in cyber, physical and systems engineering backgrounds; the models and methods used in a DT will reflect this diversity.

DT Services A DT can be understood as an integration of data, models and the services built on them. Data are acquired from external physical entities and managed within the DT; heterogeneous models capture relevant aspects of those entities and evolve under the DT's control; and services use these artefacts to deliver value, from model calibration and prediction to visualisation and stakeholder interaction. Services may be user-facing or may provide enabling capabilities for other components.

In the incubator example (Figure 3), a layered structure combines data from the PT with models of the physical plant and the software controller. These feed into *enabler* services that support the top-level services exposed to the user/operator.

Models DTs use models [43, 59] as the basis for providing services. A model is derived from a system of interest, captures a selected subset of its properties and serves a defined purpose in a given context. For DTs of CPSs, such models may describe physical or computational processes in the PT or its environment, including sensors, actuators, networks and computational platforms.

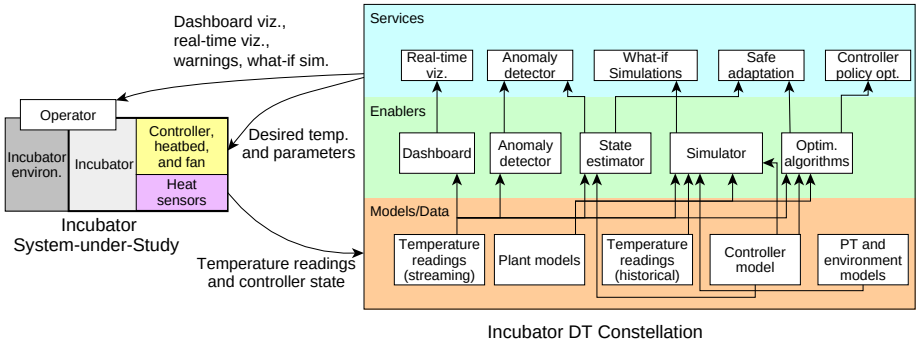


Fig. 3. Informal schematic of incubator DT showing a constellation of services. From [24].

Determining what constitutes a 'good' model for a particular purpose is important but difficult to formalise without committing to specific modelling formalisms (state machines, differential equations, Markov models, etc.). In our work, we emphasise properties such as *relevance*, *substitutability* and *fidelity*. The most critical property is *verifiability*, which is the extent to which one can determine whether the model satisfies properties of interest. Further discussion of these qualities is provided in [52].

Models can be classified by their state form (numerical or symbolic), state evolution (continuous or discrete time) and behaviour (deterministic or stochastic). These dimensions guide the choice of modelling formalism for DTs. We regard *simulation* as the process of generating a model's behaviour. *Calibration* is the process of identifying parameter values that align model behaviour with real-world data, while *validation* concerns gaining confidence in a model's qualities by comparing its predictions with observed data. Introductions to modelling can be found in [24, 11, 61]. A general treatment of simulation is in [12].

An individual DT may require multiple models capturing properties ranging from energy consumption to functional correctness. The resulting model set is necessarily heterogeneous, reflecting contributions from diverse disciplines and formalisms, including:

Physics-based models: Derived from first principles, using algebraic and (ordinary or partial) differential equations to describe physical processes such as heat transfer.

Data-driven models: Predictive models learned from input-output data, employing techniques such as machine learning, deep learning and physics-informed neural networks.

Models for computer-based systems: Using formalisms familiar in FM to describe hardware and software components, typically via discrete or hybrid models.

Model selection for a DT involves balancing factors such as precision, accuracy and the computational cost of the services that rely on model simulation or analysis.

Considering our incubator example, the DT might require a physics-based model of the temperature performance and a discrete-event model of the controller, among other models. A first principles analysis of the incubator temperature (for details see [33]) results in the following ordinary differential equation:

$$C_h \frac{\mathrm{d}T_h}{\mathrm{d}t} = H_h \cdot P_h - G_{hb} \cdot (T_h - T_b) \tag{1a}$$

$$C_b \frac{\mathrm{d}T_b}{\mathrm{d}t} = G_{hb} \cdot (T_h - T_b) - G_{br} \cdot (T_b - T_r) \tag{1b}$$

where T_h is the heater temperature, T_b is the air temperature inside the box, T_r is the known input room temperature; C_h is the heat capacity of the heater and C_b is the heat capacity of the box including contained air. The power to the heatbed P_h is a constant, while H_h is a Boolean variable switching power on/off. Finally, G_{br} is a coefficient of heat transfer between the box and the room while G_{hb} is the coefficient of heat transfer between the heater and the box.

Models of this form are termed *lumped-parameter models* because they assume the model components are concentrated in a single point, in this case assuming that the air temperature is the same throughout the full volume of the incubator. If the variation across the volume is significant for some purposes, it may be appropriate to develop a higher fidelity model based on partial differential equations that account for energy transport in fluids.

It would be natural to use a discrete state-transition model of the controller (Figure 4). This is based on bang-bang control with a slight variation. It turns on the heatbed until the temperature reaches the setpoint, then waits to prevent overshoot.

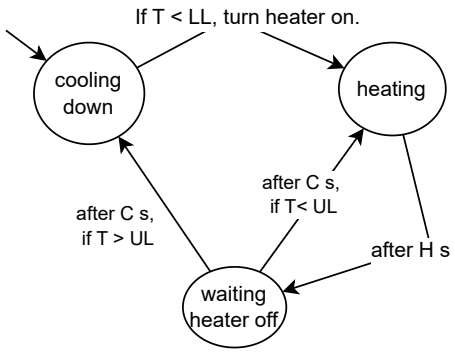


Fig. 4. Controller Statechart. LL and UL are lower and upper temperature limits; H and C are the heating and waiting durations. From [23].

Model-based DT engineering is inherently multi-disciplinary and therefore requires mechanisms for coherently linking diverse models of digital, mechanical, electrical and other processes. Much of our past work has focused on *co-simulation* [34], the coordinated simulation of an ensemble of models coupled via a standardised software interface, as a means of achieving this.

In the incubator example, co-simulation underpins decision-support services. One instance (described in detail in [33]) is a reconfiguration service triggered when the user opens the lid to inspect the incubator, invalidating the system's thermal model and controller. A monitoring service detects the resulting temperature anomaly and the reconfiguration service responds by recalibrating the thermal model and adjusting the controller to the new conditions. It selects a lower temperature setpoint to save energy and avoid the overshoot that may occur when the lid is closed again [64]. Figure 5 illustrates this process, marking the moments when the lid is opened and closed and showing how the controller profile adapts accordingly.

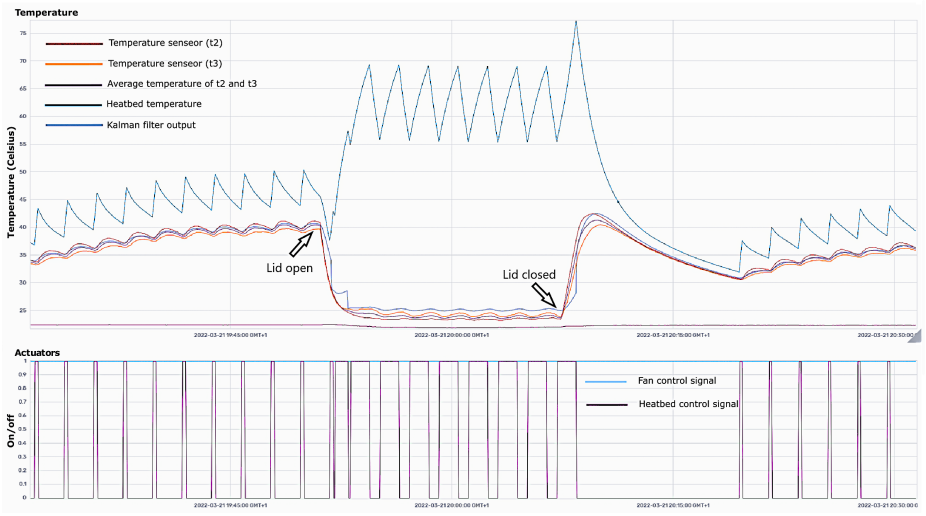


Fig. 5. Experimental result of self-adaptation. Adapted from [22].

This part of the tutorial has introduced core DT concepts and building blocks, illustrated through an incubator study which is available as an open proof-of-concept [33]. We have highlighted the central role of models and the services that exploit them during operational life.

4 Examples of FM Applications in DT Engineering

DT engineering requires methods, tools and processes that can handle the plurality and heterogeneity of models and services that constitute a DT. They

must also support maintaining convergence between PT and DT, recalibration and reconfiguration. This section outlines three recent studies at Aarhus University that address these challenges. The topics include dynamic management of updates in a live DT-enabled system, the role of runtime monitoring and verification, and support for DT construction.

4.1 Concrete Batch Plant

This case study concerns **fibo Collect**, a concrete batch plant developed by **Fibo Intercon**. The plant is a fully automated system responsible for handling, batching and mixing stone, sand, water and cement to produce certified concrete in production environments. The plant operates continuously and customers demand high availability: interruptions or incorrect behaviour can cause production losses and material waste.

As part of its ongoing evolution, fibo Collect is being equipped with a DT to enable closer integration between the physical plant and its digital counterpart. The resulting DT-enabled system is intended to enhance monitoring, support optimisation, facilitate anomaly detection and ultimately increase plant uptime. However, introducing a DT also brings new engineering challenges, most notably the need to update software services safely in a live DT-enabled system.



Fig. 6. fibo Collect concrete batching plant, at Fibo Intercon A/S. Image credit: Fibo Intercon A/S, used with permission.

Because fibo Collect requires a human operator to supervise the system during updates, it is not feasible to perform updates while the plant is offline at night. Instead, new service versions must be deployed while the system remains operational and the transition from old to new services must preserve the plant's availability. These challenges are explored in two complementary works: a probabilistic update scheduler (PUS) presented in [3] and an as-yet unpublished study applying model checking to a swap algorithm for stateful services.

Probabilistic Update Scheduler These challenges led us to hypothesise that DT-enabled systems evolve through primary states that can be classified as either *safe* or *unsafe*. This perspective gives rise to three key problems: (i) identifying when the DT-enabled system is in a safe state; (ii) determining the duration of each safe state; and (iii) predicting when the system will next enter a safe state.

To tackle these issues, we model each component of the DT-enabled system as a Semi-Markov process (SMP), capturing both its state transitions and associated sojourn times. The SMPs are defined as follows. First, the probability that the chain moves from state i to state j after $n+1$ transitions is given in eq. (2) and the wait time distribution, adapted from Medhi [49] is given by eq. (3).

$$Q_{ij}(n) = P(X_{n+1} = j \mid X_n = i) \tag{2}$$

$$W_{ij}(t) = P(T_{n+1} - T_n \leq t \mid X_{n+1} = j, X_n = i) \tag{3}$$

Using the above equations, we can model multiple SMPs corresponding to different parts of the DT-enabled system. This allows us to predict both the current and future behavioural states of the system and to compute the probability that it remains in a safe state over the current time and the next n transitions. To quantify *safety*, a Monte Carlo Simulation approach is used. Starting from an initial state, the system can be simulated for a fixed target duration (e.g., 30 minutes per simulation) across a large number of independent runs. For each run, it is determined if the system remained in a safe state for the target duration. The probability that the system remains in the set of safe states of $\mathcal{S}$ is estimated as the proportion of simulation runs where all visited states are safe together at the target duration. This probability enables us to assess the confidence in the scheduled update time and determine whether or not a customer should be informed.

Simulating the collection of SMPs yields the probability that the system is currently in a safe state, as well as its likelihood of remaining safe in the future. This information can guide decisions about when to update a DT-enabled system, reducing the risk of performing updates at inappropriate times.

Future work includes the automated verification of such probabilistic models, i.e., checking whether the system satisfies given probabilistic properties. For example, we may wish to verify that the probability of eventually reaching a state in which a service has been successfully swapped meets a required threshold. Using *Probabilistic Computation Tree Logic* (PCTL), this can be expressed as a property such as "eventually a service will be swapped with probability at least 98%," formalised as shown in eq. (4).

$$P_{\geq 0.98}\left[\, \Diamond \, \mathit{ServiceSwapped} \,\right] \tag{4}$$

Swapping stateful services While the PUS determines *when* an update attempt may take place to reduce the risk of system interruptions, it does not address *how* a running DT service can be replaced without disrupting the operation of other DT services or the PT. DT services are often stateful, meaning

they cannot be stopped or restarted without losing information, which may lead to errors or system-wide disruptions. To support updates without taking the system offline, we require a method for determining when two instances of the same DT service are behaviourally equivalent. We therefore focused on comparing the input-output behaviour of the existing DT service with that of the candidate replacement. Once the replacement service produced outputs equivalent to those of the original for the same inputs, the system would perform a swap.

To build confidence in this approach, the correctness of the swapping mechanism was specified in PlusCal and translated into TLA+, enabling us to model the services and the swap mechanism. We then used the TLC model checker to verify that the system satisfied the intended swapping properties. These included a range of safety and liveness conditions, among them the liveness property shown in eq. (5), which asserts that the system should eventually perform the service swap.

$$Liveness \triangleq \Diamond\Box(swapped \Rightarrow swapped') \tag{5}$$

This case study highlights opportunities for integrating FMs into operational DT settings. By combining probabilistic reasoning for update scheduling with a verified mechanism for swapping stateful services, the approach offers a path toward more reliable continuous deployment. Future work includes implementing the formally verified swapping mechanism in the concrete batch plant and deploying it in production, allowing us to further assess its effectiveness in increasing deployment frequency within a DevOps workflow.

4.2 Runtime Verification and an Autonomous Mobile Robot

Digital Twins and Runtime Verification One advantage of DTs is their ability to support continuous system monitoring, enabling the detection and reporting of anomalies at runtime based on real operational data. Such capabilities provide stakeholders with critical information that can help prevent failures or inform recovery strategies. Monitoring services can also be combined with complementary DT functionalities, such as visualisation, further increasing the value derived from the DT. Research opportunities related to runtime monitoring of DTs, including validity frame monitoring, are discussed in Section 5.

Although monitoring services can, in principle, be implemented in any programming language, such ad hoc implementations often lack the precision required for complex or critical applications. A formal approach to monitoring offers clear advantages: it enables the specification of precise, unambiguous system properties with mathematically defined semantics over execution traces. This perspective underpins the field of *Runtime Verification (RV)*, which investigates methods and tools for checking system behaviour at runtime to increase confidence in correctness and reliability.

In the context of DTs, RV allows a designer to write a formal specification that is then used to evaluate the observed behaviour of the DT against the

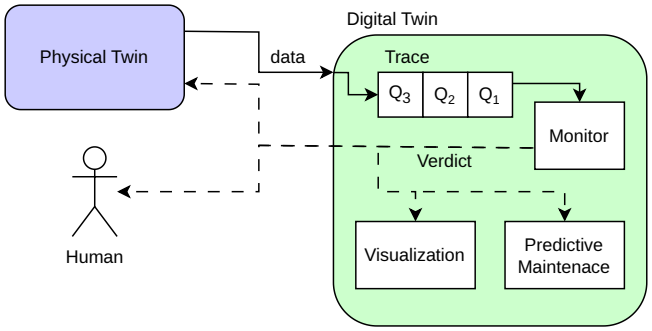


Fig. 7. The basic idea of monitoring within a DT scenario. "Visualization" and "Predictive Maintenance" are examples of other DT services.

expected properties. The monitor issues verdicts that may be used by the PT, by other DT services, or by a human-in-the-loop, as illustrated in Figure 7.

The introduction of the Lola language marked the emergence of a new RV paradigm known as *Stream Runtime Verification (SRV)* [16]. In SRV, declaratively specified output streams of verdicts are computed from input streams via stream equations. Unlike traditional RV languages rooted in temporal logics, SRV formalisms draw inspiration from functional reactive programming, providing a compositional and declarative framework for defining efficient monitors capable of continuously processing and reasoning about real-time data streams.

Lola is a minimal SRV language and most subsequent SRV formalisms implement supersets of its core *stream expressions*. These include constant-valued streams, arithmetic and boolean operators, conditional if–then–else expressions and temporal offset operators. Notable developments in this line of work include RTLola [21], TeSSLa [46] and hLola [13].

In this paper, we adopt the recent language DSRV (pronounced "Deserve"), previously introduced as DynSRV in [42]. Examples of DSRV properties are shown below, where $\texttt{default}(z[N], V)$ returns the Nth previous value of stream z, or V if that value is unavailable. Equation (6) illustrates basic conditional and arithmetic constructs of the language.

Equations (7) and (8) demonstrate counters, where (7) increments z by 1 indefinitely as is it not bound by any input streams and (8) increments z by x when a new x-value is received. Equations (9) and (10) demonstrate the implementation of a moving-average filter with $M = 3$, where (10) defines it recursively.

$$z = \texttt{if } x \leq 42 \texttt{ then } \text{true } \texttt{else } y \cdot 2 \leq x \tag{6}$$

$$z = \texttt{default}(z[1], 0) + 1 \tag{7}$$

$$z = \texttt{default}(z[1], 0) + x \tag{8}$$

$$z = \frac{(x + \texttt{default}(x[1], 0) + \texttt{default}(x[2], 0))}{3} \tag{9}$$

$$z = \frac{\texttt{default}(z[1], 0) + (x - \texttt{default}(x[3], 0))}{3} \tag{10}$$

Systems often evolve over time due to changing environments, self-adaptation mechanisms, or other dynamic factors [8] and the RV monitor may need to evolve accordingly. In some situations, restarting the monitor with an updated specification may be acceptable, even though this entails losing its internal state. In other cases, particularly when continuous monitoring is essential, such restarts are unacceptable, as they risk producing incorrect verdicts due to the loss of the complete execution trace. Consider, for example, a property requiring that a robot undergo maintenance at least once every 24 hours. If the monitor is restarted 20 hours after the last maintenance (thereby losing this timestamp) and the next maintenance occurs 10 hours later, the resulting 30-hour gap constitutes a violation that would go undetected.

To address this challenge, DSRV introduces *Dynamically Updated Properties* (DUPs), which allow monitoring properties to evolve at runtime without restarting the monitor. The DUP primitive $\texttt{dynamic}(p)$ evaluates a stream expression p dynamically within the current trace context. As illustrated in eq. (11), this can be combined with static properties via conjunction: the left-hand side enforces the constraint $T < T_m$ (the time since last maintenance, T, must remain below the fixed threshold T_m), while the right-hand side strengthens this requirement with a dynamic constraint derived from p, defaulting to true when unavailable.

Notably, conjunction permits strengthening, e.g., requiring more frequent maintenance during mission-critical phases, but does not permit relaxation of the window. For a detailed discussion of DUPs and associated design patterns in DSRV, we refer to [42].

$$z = T < T_m \wedge \texttt{default}(T < \texttt{dynamic}(p), \text{true}) \tag{11}$$

Digital Twin of Autonomous Mobile Robot We demonstrate the use of DSRV in a scenario featuring an Autonomous Mobile Robot (AMR) operating in a warehouse with a DT. The DT provides a visualisation service through which AMR metrics can be inspected and new navigation goals can be set. The AMR requires periodic recharging, which is autonomously managed by a decision-making service within the DT. According to the datasheet, the battery supports at least 2.5 hours of operation; the service therefore maximises uptime by initiating recharging as late as possible without risking full depletion.

However, reliably determining when the robot must be scheduled to recharge is non-trivial. Dynamic task priorities, environmental changes (e.g., blocked pathways) and uncertainty in travel-time prediction all contribute to this complexity. To increase confidence in the system's behaviour, the DT continuously monitors the property in eq. (11). Here, T is an input stream that increments every minute and resets after each recharge, $T_m = 150$ enforces recharging at least every 2.5 hours, p provides dynamic updates from DT services or human operators and z is the resulting verdict stream.

The following three scenarios illustrate how this property evolves in tandem with the DT services.

Scenario 1: Trends in the visualisation service reveal that the AMR's battery frequently reaches critical levels during recharge attempts. To mitigate the risk of failing to reach the charging station, operators reconfigure the DT service to schedule recharges every 2 hours as a temporary measure. An operator updates the monitor by sending $p = 120$, essentially reducing eq. (11) to the stricter property $z = T < 120$ until further updates to p are received.

Scenario 2: As a more robust solution, the DT introduces a battery model predicting capacity degradation over time, enabling dynamic T_m adjustment based on projected remaining life. The monitor updates via $p = d$, where d is an input stream from the battery model service carrying expected lifetime values. This strengthens the property to $z = T < T_m \wedge T < d$, ensuring maintenance aligns with actual battery health.

Scenario 3: As trust in the AMR grows, it is assigned increasingly mission-critical tasks (e.g., high-priority orders) within the warehouse, elevating the consequences of battery failure. The DT is updated to incorporate mission-criticality streams c from the task planner and the monitor is updated accordingly via $p = \texttt{if}\ c\ \texttt{then}\ d - 30\ \texttt{else}\ d$, to tighten recharge requirements during critical missions. Alternatively, for greater generality, $p = \texttt{if}\ c\ \texttt{then}\ \texttt{defer}(p[1]) - 30\ \texttt{else}\ \texttt{defer}(p[1])$ updates the dynamic property by reusing the previously assigned value of p via the `defer` primitive. `defer` is a DUP primitive similar to `dynamic` except it only allows dynamically updating a property once.

Our work with DSRV demonstrates how DUPs enable dynamic reconfiguration of monitoring properties in DTs, allowing seamless adaptation without monitor restarts. Several open questions remain:

1. *Practical deployment of DUPs* in real-world settings such as self-adaptive systems and dynamic software updating environments, where specifications must evolve alongside system behaviour.
2. *Formal verification of DUP safety*, in which static meta-properties within a DSRV specification ensure that dynamically introduced properties satisfy required constraints, thereby restricting the space of admissible updates.
3. *Integration with complementary FMs*, using theorem provers or model checkers to verify that dynamically introduced properties preserve overall specification integrity and critical constraints.

This case study illustrates how SRV can support robust and adaptable runtime monitoring in DT-enabled systems. By enabling properties to evolve without interrupting monitoring, DSRV gives a basis for handling dynamic operational contexts. At the same time, the open challenges identified above highlight opportunities for deeper integration between SRV and other FMs.

4.3 Coupling UR and Kuka Robots

It is challenging to engineer DTs for robotic systems, in part because robotics relies on a wide range of modelling principles and methods [58], including kinematics, dynamics, geometry, forces, control and other domain-specific aspects. The complexity increases further in scenarios involving robot–robot cooperation or human–robot collaboration, where multi-system interactions and heterogeneity must be addressed. Compounding this, specialised tooling for DT engineering in robotics remains limited [48], making the process more time- and resource-intensive. Nevertheless, the robotics community already offers several tools that can support DT development, including the Robot Operating System (ROS), which has been widely adopted for this purpose [57, 47], as well as robotic simulators such as Gazebo [47], CoppeliaSim [56] and the Robotics Toolbox [15].

For this example, we consider the Flex-cell (see Figure 8), a manufacturing cell composed of two robotic arms: a UR5e and a Kuka LBR iiwa 7. Although both arms belong to the same system and appear similar, they differ in their underlying models, control methods, communication and software interfaces, and crucially their simulation environments. As a result, they must be treated as two independent and heterogeneous sub-systems within the Flex-cell case study.

This heterogeneity has made the Flex-cell a useful platform for demonstrating a range of DT engineering methods, including modelling composable DTs for cooperative robotic systems [29], integrating the *robot skills* concept [54] in DTs [32], describing and reporting engineered DTs [31], executing simulations of coupled heterogeneous systems [27] and engineering DTs that combine co-simulation with RoboStar technologies [9, 26].

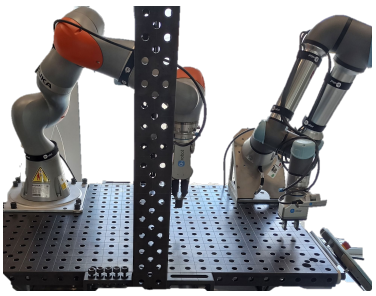


Fig. 8. Flex-cell manufacturing cell. From [27].

When engineering a DT for a composed system with heterogeneous components like the Flex-cell, it is important to consider not only the system-wide aspects, but also the characteristics of each internal subsystem, so the system as a whole behaves as intended. From our experience with this example, several system-level considerations are fundamental to fine-tuning composed DTs, including interoperability and common semantics, co-orchestration of components, common and reusable interfaces and time synchronisation. To address these aspects, we have used ontologies as common vocabularies and structure/knowledge graphs, communication protocols such as MQTT and AMQP that support reusable and scalable patterns, and co-simulation to deliver coordinated execution and time synchronisation.

Intra sub-system aspects are also important to consider when engineering the system as a whole. Among the publications using the Flex-cell as a case study, the work combining co-simulation and RoboStar technologies [26], provides the most detailed treatment of the internal aspects of each of each subsystem. RoboStar offers a unified robotics toolkit that disaggregates the representation of a robotic platform into several formally specified components, including the physical modelling (*p-model*), the control software modelling (*d-model*), the mapping between the control software and the physics (*platform mapping*), and the world assumptions (*RoboWorld*). A key benefit brought by RoboSim [10], one of RoboStar's tools, is support for automatic validation and verification of model properties, and automatic code generation (compliant with the properties already verified).

In [26], co-simulation was used to orchestrate the components at the subsystem level with finer granularity, following a pattern established in earlier Flex-cell studies. More recently, this has been extended with the formalism of *synchronous clocks* to provide better manoeuvrability on both axes of time (continuous-time axis and discrete-event axis) and to improve the observability of state machine computations in the co-simulation of robotic systems [30].

Overall, the Flex-cell case study shows that coupling heterogeneous robotic sub-systems demands attention to both system-level coordination and detailed sub-system modelling, co-simulation and shared semantics. Future work for the Flex-cell include a more detailed design of the cooperative scenarios (covering both the intra sub-system and macro system aspects with a wider set of models with higher fidelity) and extending its capabilities to multi-robot cooperation with other robotic systems and human-robot collaboration.

5 Opportunities for Formal Methods

Our prior work [24] and recent experience (e.g., Section 4) suggests that there are many opportunities for FMs research and innovation, both in the construction of dependable DTeSs and within operational DTs. The recent report from the US National Academies of Science, Engineering and Medicine [51] emphasises the need for Verification, Validation and Uncertainty Quantification as a continual process. This gives rise to numerous opportunities for the FMs. We here identify areas where we see the highest level of opportunity for the FM community.

5.1 The Reality Gap, Validity Frames and Uncertainty

There is an unavoidable *reality gap* between a real-world CPS and its representation in a DT. Consequently, when decisions and services depend on DT-based representations, it is essential to characterise and understand the bounds on their accuracy. These bounds are shaped by uncertainties in sensing, latency, modelling and environmental variability. Idealised assumptions—such as noise-free sensors—are often adopted to keep analysis tractable, but recognising such assumptions is crucial for assessing the limitations of DT predictions and their impact on decision-making. The FM community is well positioned to characterise these limits systematically, enabling more principled reasoning about confidence in DT-based analyses.

A further challenge is determining whether DT models can adapt to environmental changes not anticipated when the PT was originally engineered. This relates closely to the notion of *validity frames*, which specify the conditions under which a DT can be expected to provide acceptable predictions [1]. FMs have significant potential to support the definition and analysis of validity frames, offering rigorous guarantees about the conditions under which DT predictions remain dependable.

Techniques for configuring and propagating uncertainty in CPS models have been studied for many years. Continued research is needed at the DT level to address the challenges outlined above. Different application domains employ distinct techniques for representing and propagating uncertainty, including reachability analysis [19], Monte-Carlo simulation [63], stochastic differential equations [5] and sensitivity analysis [14]. Formal techniques can help provide precise representations of uncertainty within DT models, define semantics for how uncertainty evolves across DT layers and offer guarantees about the correctness or robustness of DT-based predictions.

5.2 Assurance at Runtime

As DT-enabled systems increasingly support monitoring, decision-making and autonomous adaptation during operation, confidence in their behaviour can no longer rest solely on offline verification. Instead, assurance must be maintained continuously, in the presence of evolving models, uncertain data and potentially hostile environments. This shifts the focus from static guarantees to runtime assurance mechanisms that combine monitoring, analysis and adaptation. Formal methods provide a natural foundation for such mechanisms, offering precise semantics for observed behaviour, principled ways to detect and respond to deviations, and ways to reason rigorously about about safety and correctness as systems evolve in operation.

Towards Verified DTs DTs are proposed as enablers for self-adaptive CPSs that can modify their behaviour in response to evolving and previously unseen environments. While existing work has explored architectures and adaptation mechanisms, there has been limited progress on providing guarantees for such

runtime adaptations. In particular, there is a need for formal foundations, semantics and concepts for RV (as discussed in Section 4.2) and monitoring in DT-enabled systems [64, 42].

A key challenge in this setting is the development of integrated formal foundations for CPSs that can underpin DT services supporting decision-making, particularly under high uncertainty. This requires unifying frameworks able to accommodate diverse verification tools: provers such as Isabelle [53] and Coq [6] for discrete systems; tools such as KeYmaera X [55] for hybrid models; and model checkers such as Storm [18] for probabilistic behaviour. Such foundations would enable a range of use cases, including probabilistic DTs augmented with Markov and Bayesian models, probabilistic risk models to support operational decisions and the verification of DTs within certification or compliance processes.

Security of DT-enabled Systems Security is a particular concern for DT verification. The communications infrastructure of a DT-enabled system—including sensors, actuators, networks and the DT itself—creates opportunities for malicious actors to disrupt the PT [2, 62]. It is therefore essential to consider how to protect DT-enabled systems from such attacks [45]. While specific countermeasures depend on the deployment context, systematic exploration of attack patterns and mitigation strategies is required [44]. As DT deployments scale to Digital Twin as a Service (DTaaS) platforms hosting many DTs, it becomes critical to ensure strong isolation and protection of each DT's data space.

These challenges present clear opportunities for FMs. Potential contributions include identifying vulnerabilities and exploit paths in DT-enabled systems; developing and specialising FMs and tools for verifying DT security; cataloguing DT-specific attack models and mitigation strategies; and designing low-overhead, formally justified security protocols for bidirectional DT–PT communication.

5.3 Engineering Automation & Evolution

A defining characteristic of DTs is that, over time, they yield new engineering knowledge about how a PT operates in its actual deployment environment, often in ways that were not anticipated at design time. Exploiting this knowledge to optimise DT services for a specific PT naturally motivates evolution, both of models and of the services built upon them. However, experience shows that such evolution is hindered by limited automation in DT production, a lack of modelling languages with precise semantic foundations and insufficient support for evolving DT-enabled systems safely while they remain operational. These challenges are tightly coupled: automation depends on sound language foundations, while safe evolution presupposes both automated reasoning and explicit models of system structure and behaviour. Together, they define a core set of engineering problems where FMs can play a decisive role.

Automating Digital Twin Production Creating a dependable DT (and DTeS) from scratch is resource-intensive. Reusable, model-based frameworks

are therefore essential for systematic DT engineering. Such frameworks require a well-defined architectural core and a coherent set of language concepts and services, enabling the integration of heterogeneous models that can be adapted and extended for specific DT instances. Precise semantic foundations are needed to ensure that reuse and extension preserve correctness and intended system properties.

Most existing DT frameworks inherit heavily from Internet of Things (IoT) architectures, which focus primarily on data communication between the PT and the DT. Consequently, automation is largely limited to data acquisition and synchronisation [28], with far less support for automated model-based simulation or formal checking within runtime monitors. Providing comparable automation for simulation, analysis and verification would significantly strengthen assurance and enable DT frameworks that move beyond data-centric integration.

Modelling Languages for Digital Twins Theories and techniques for combining and managing heterogeneous languages are well established in software engineering, yet their systematic application to DT frameworks remains limited. Moyne et al. identify medium- and long-term requirements for DT frameworks, including dedicated languages for DT definition, support for reuse and portability, interoperability and interchangeability of DTs, and integrated validation and verification capabilities [50].

Languages such as the Digital Twin Definition Language (DTDL)[5], an open-source language targeting Azure DTs, offer a pragmatic starting point for automating aspects of DT construction and are being extended with features such as configuration data and services. However, such languages typically lack a formally defined semantics, limiting rigorous analysis and verification. This gap highlights clear opportunities for FMs to provide semantic foundations and enable automated reasoning within DT engineering.

Safe Evolution during Operation Synchronisation between the PT and DT is fundamental to ensuring that the DT accurately reflects the evolving state of the PT. At the same time, functions such as control, anomaly detection and behavioural optimisation depend on timely information flowing from the DT back to the PT. Achieving scalable, bi-directional communication and synchronisation throughout the lifecycle of DT-enabled systems remains a significant challenge, particularly under fault conditions where increased PT error rates can raise data traffic and latency, potentially undermining the feasibility of interventions. This creates a clear opportunity for the FM community to contribute principled, formally justified communication and synchronisation protocols that minimise information exchange without compromising correctness or responsiveness [60].

A related challenge concerns the automated evolution of software within DT-enabled systems, especially in settings that require the safe deployment of updates while systems remain in use, consistent with DevOps practices. In particular, over-the-air updates should only be applied when the PT is in a state

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

that guarantees safe installation and minimal disruption to its primary function. FMs can play a key role here by enabling rigorous analysis and planning of update strategies, including identifying optimal update points that balance availability, safety and operational efficiency [3].

5.4 The Ecosystem of Models and Digital Twins

A model's abstraction level is determined by its intended use case and the system properties it must reproduce, so engineering a complex CPS typically produces multiple models spanning different viewpoints, assumptions and development stages. DTs create an opportunity to exploit this diversity at runtime, though doing so remains challenging. As DT services evolve and new knowledge of the PT emerges, different models — or heterogeneous combinations such as those in Section 4.3 — may become preferable. Model choice depends not only on original intent but also on whether it can align with live data and operate as a digital shadow. Establishing principled methods to manage and combine models in this dynamic setting is a key challenge for FMs.

Leveraging Multiple Levels of Abstraction and Fidelity Managing abstraction is a familiar concern in formal methods, where modelling choices are guided by a model's intended purpose and the properties it must reproduce. As a result, multiple models of the same system are often developed, each reflecting a different engineering viewpoint. A DT provides an opportunity to exploit these diverse models during operation as DT services evolve, more is learned about the PT's deployed environment and stakeholder needs change.

This creates the challenge of selecting — potentially autonomously — the most appropriate model, based not only on its purpose but also on its performance, fidelity and our ability to ensure that it remains a faithful representation of the PT [40]. Recent work has explored dynamic model changes [20], including replacing models and altering the DT's structure (i.e., the interactions between models) while maintaining safety guarantees. Some control here may be data-driven, making it essential to ensure safety under data-dependent adaptation [7].

FMs have a clear role to play in analysing how models are characterised and selected and in providing assurance that model switching remains safe over time. A key challenge lies in formalising notions of abstraction and fidelity—concepts that are inherently subtle but whose effects propagate directly to DT services.

Capture and Representation of Causal Relations The reality gap arises from misalignment between the PT, its environment and their DT representations, motivating the need for formal representations of system operating modes and precise characterisations of the conditions under which transitions between such modes occur. Established formalisms such as Hybrid Automata [25] and Hidden Markov Models [4], together with their underlying theoretical foundations, provide natural starting points for this endeavour.

From a FMs perspective, a key challenge is distinguishing mere correlations from genuine causal relations between system states and observed behaviours. Addressing this may require a principled integration of formal reasoning with the design and analysis of experiments, enabling the formulation, validation and refutation of explicit causal hypotheses.

Enabling Interoperability and Composition To date, only limited attention has been paid to the composition of DTs or DTeSs. A recent review [41] identifies a spectrum of composition approaches, ranging from loosely coupled DTs that interact through defined interfaces to highly centralised orchestrations. However, when it comes to validating and verifying such compositions, most studies rely on simulation-based or qualitative assessments, with comparatively little focus on scalable formal frameworks.

Full automation of interoperability between DTs presupposes agreed standards for DT interfaces. At present, however, standards, frameworks and regulations specific to DTs remain underdeveloped [17]. In practice, standards tend to emerge from established communities of developers and users, and such communities are only beginning to form. For example, the Digital Twin Consortium[6] operates within the Object Management Group and represents a step toward coordinated efforts. These initiatives must be sustained and extended across different technology stacks so that competition centres on functionality and quality of DT services rather than proprietary choices or alliances. In this regard, the development of SysML Version 2[7] marks significant progress for Model-Based Systems Engineering (MBSE) and offers a promising foundation for future DT-related standardisation.

Other standardisation efforts are underway, but such initiatives are inherently slow, requiring alignment of baseline concepts. As a result, it will take time before new research results can be incorporated into emerging standards. Moreover, interoperability between DTs remains an under-emphasised aspect of current work. Further research is needed at a cognitive layer to understand how collaboration between DTs can be effectively enabled.

6 Conclusions

This tutorial has introduced the fundamentals of DT technology, highlighting opportunities for FMs to advance research and innovation in this rapidly developing field. We have emphasised the central role of models, the added value of services built upon them, and the challenges of maintaining alignment with evolving PTs and environments. Three recent case studies have illustrated the importance of dynamic model updates, the role of runtime verification and the challenges of linking diverse PTs.

[6] `https://www.digitaltwinconsortium.org/`
[7] `https://github.com/Systems-Modeling/SysML-v2-Release`

We have outlined several research challenges where FMs can make a meaningful contribution. These include understanding the characteristics and applicability limits of the heterogeneous models that constitute a DT, providing assurance at runtime, engineering dependable DTs and exploiting the emerging ecosystem of models and DTs. In many cases, the key contribution lies not in inventing new formalisms, but in developing principled ways to harness existing methods and tools coherently and soundly.

DT technology offers a rare opportunity to place FMs at the heart of digital transformation, enabling the development of systems that are not only more capable, but also more dependable and trustworthy. As the field continues to grow, we hope this tutorial serves as both an entry point and an invitation for the community to engage with the rich and expanding landscape of DT research.

Acknowledgments. We are grateful to many colleagues at Aarhus and Newcastle Universities for supporting our work to date, as well as to numerous agencies funding our DT research but in particular the Grundfos Foundation for establishing the AU Centre for Digital Twins and the DIGIT BENCH project for funding a part of this research. **Disclosure of Interests.** The authors have no competing interests to declare.

References

1. Acker, B.V., Meulenaere, P.D., Vangheluwe, H., Denil, J.: Validity frame–enabled model-based engineering processes. Simulation (2023). https://doi.org/10.1177/00375497231205035
2. Alcaraz, C., Lopez, J.: Digital twin: A comprehensive survey of security threats. IEEE Communications Surveys & Tutorials **24**(3), 1475–1503 (2022). https://doi.org/10.1109/COMST.2022.3171465
3. Andersen, M.S., Gomes, C., Thompson, S., Larsen, P.G.: Probabilistic Update Scheduling for Digital Twins: A Semi-Markov Approach. In: 2025 ACM/IEEE 28th Intl. Conf. on Model Driven Engineering Languages and Systems Companion (MODELS-C). pp. 250–256. IEEE Computer Society (Oct 2025). https://doi.org/10.1109/MODELS-C68889.2025.00043
4. Baum, L.E., Petrie, T.: Statistical inference for probabilistic functions of finite state Markov chains. Annals of Mathematical Statistics **37**(6), 1554–1563 (1966)
5. Bayram, M., Partal, T., Buyukoz, G.O.: Numerical methods for simulation of stochastic differential equations. Advances in Difference Equations **2018**(17) (January 2018). https://doi.org/10.1186/s13662-018-1466-5
6. Bertot, Y., Casteran, P.: Interactive Theorem Proving and Program Development. Springer Verlag (2004). https://doi.org/10.1007/978-3-662-07964-5
7. Brunke, L., Greeff, M., Hall, A.W., Yuan, Z., Zhou, S., Panerati, J., Schoellig, A.P.: Safe Learning in Robotics: From Learning-Based Control to Safe Reinforcement Learning. Annual Review Control, Robotics, and Autonomous Systems **5**, 411–444 (May 2022). https://doi.org/10.1146/annurev-control-042920-020211
8. Carwehl, M., Vogel, T., Rodrigues, G.N., Grunske, L.: Runtime Verification of Self-Adaptive Systems with Changing Requirements. In: IEEE/ACM Symp Software Engineering for Adaptive and Self-Managing Systems. pp. 104–114 (2023). https://doi.org/10.1109/SEAMS59076.2023.00024

9. Cavalcanti, A., Barnett, W., Baxter, J., Carvalho, G., Filho, M.C., Miyazawa, A., Ribeiro, P., Sampaio, A.: Robostar technology: A roboticist's toolbox for combined proof, simulation, and testing. In: Cavalcanti, A., Dongol, B., Hierons, R., Timmis, J., Woodcock, J. (eds.) Software Engineering for Robotics, pp. 249–293. Springer (2021). https://doi.org/10.1007/978-3-030-66494-7_9
10. Cavalcanti, A., Sampaio, A., Miyazawa, A., Ribeiro, P., Conserva Filho, M., Didier, A., Li, W., Timmis, J.: Verified simulation for robotics. Science of Computer Programming **174**, 1–37 (2019). https://doi.org/10.1016/j.scico.2019.01.004
11. Cellier, F.E.: Continuous System Modeling. Springer New York (1991). https://doi.org/10.1007/978-1-4757-3922-0
12. Cellier, F.E., Kofman, E.: Continuous System Simulation. Springer New York (2006). https://doi.org/10.1007/0-387-30260-3
13. Ceresa, M., Gorostiaga, F., Sánchez, C.: Declarative Stream Runtime Verification (hLola). In: Oliveira, B.C.d.S. (ed.) Programming Languages and Systems. pp. 25–43. Springer (2020). https://doi.org/10.1007/978-3-030-64437-6_2
14. Chou, Y., Sankaranarayanan, S.: Bayesian parameter estimation for nonlinear dynamics using sensitivity analysis. In: Proc. 28th Intl. Jnt. Conf. on Artificial Intelligence, IJCAI-19. pp. 5708–5714. Intl. Jnt. Confs. on Artificial Intelligence Organization (7 2019). https://doi.org/10.24963/ijcai.2019/791
15. Corke, P., Haviland, J.: Not your grandmother's toolbox – the robotics toolbox reinvented for python. In: 2021 IEEE Intl. Conf. on Robotics and Automation (ICRA). pp. 11357–11363 (2021). https://doi.org/10.1109/ICRA48506.2021.9561366
16. D'Angelo, B., Sankaranarayanan, S., Sanchez, C., Robinson, W., Finkbeiner, B., Sipma, H., Mehrotra, S., Manna, Z.: LOLA: Runtime Monitoring of Synchronous Systems. In: Proc. Intl. Symp. on Temporal Representation and Reasoning. pp. 166–174 (2005). https://doi.org/10.1109/TIME.2005.26
17. David, I., Shao, G., Gomes, C., Tilbury, D., Zarkout, B.: Interoperability of Digital Twins: Challenges, Success Factors, and Future Research Directions, pp. 27–46. Springer (Oct 2024). https://doi.org/10.1007/978-3-031-75390-9_3
18. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A Storm is Coming: A Modern Probabilistic Model Checker. In: Majumdar, R., Kunčak, V. (eds.) Computer Aided Verification. pp. 592–600. Springer (2017). https://doi.org/10.1007/978-3-319-63390-9_31
19. Donzé, A.: Breach, a toolbox for verification and parameter synthesis of hybrid systems. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification. pp. 167–170. Springer (2010). https://doi.org/10.1007/978-3-642-14295-6_17
20. Ejersbo, H., Lausdahl, K., Frasheri, M., Esterle, L.: Dynamic Runtime Integration of New Models in Digital Twins. In: 2023 IEEE/ACM 18th Symp. on Software Engineering for Adaptive and Self-Managing Systems (SEAMS). pp. 44–55 (2023). https://doi.org/10.1109/SEAMS59076.2023.00016
21. Faymonville, P., Finkbeiner, B., Schledjewski, M., Schwenger, M., Stenger, M., Tentrup, L., Torfah, H.: StreamLAB: Stream-based Monitoring of Cyber-Physical Systems. In: Dillig, I., Tasiran, S. (eds.) Computer Aided Verification. pp. 421–431. Springer (2019). https://doi.org/10.1007/978-3-030-25540-4_24
22. Feng, H., Gomes, C., Gil, S., Mikkelsen, P.H., Tola, D., Larsen, P.G., Sandberg, M.: Integration Of The Mape-K Loop In Digital Twins. In: 2022 Annual Modeling and Simulation Conference (ANNSIM). pp. 102–113. IEEE, San Diego, CA, USA (Jul 2022). https://doi.org/10.23919/ANNSIM55834.2022.9859489

23. Feng, H., Gomes, C., Thule, C., Lausdahl, K., Sandberg, M., Larsen, P.G.: The incubator case study for digital twin engineering (2021). https://doi.org/10.48550/arXiv.2102.10390
24. Fitzgerald, J., Gomes, C., Larsen, P.G. (eds.): The Engineering of Digital Twins. Springer (2024). https://doi.org/10.1007/978-3-031-66719-0
25. Frehse, G.: An Introduction to Hybrid Automata, Numerical Simulation and Reachability Analysis. In: Drechsler, R., Kühne, U. (eds.) Formal Modeling and Verification of Cyber-Physical Systems: 1st Intl. Summer School on Methods and Tools for the Design of Digital Systems, Bremen, Germany, September 2015. pp. 50–81. Springer, Wiesbaden (2015). https://doi.org/10.1007/978-3-658-09994-7_3
26. Gil, S., Badyal, A., Miyazawa, A., Larsen, P.G., Cavalcanti, A.: A model-based approach for co-simulation-driven digital twins in robotics. Robotics and Autonomous Systems **196**, 105240 (2026). https://doi.org/10.1016/j.robot.2025.105240
27. Gil, S., Kamburjan, E., Talasila, P., Larsen, P.G.: An architecture for coupled digital twins with semantic lifting. Software and Systems Modeling **24**(5), 1379–1404 (2025). https://doi.org/10.1007/s10270-024-01221-w
28. Gil, S., Mikkelsen, P.H., Gomes, C., Larsen, P.G.: Survey on open-source digital twin frameworks–A case study approach. Software: Practice and Experience **54**(6), 929–960 (2024). https://doi.org/10.1002/spe.3305
29. Gil, S., Mikkelsen, P.H., Tola, D., Schou, C., Larsen, P.G.: A Modeling Approach for Composed Digital Twins in Cooperative Systems. In: 2023 IEEE 28th Intl. Conf. on Emerging Technologies and Factory Automation (ETFA). pp. 1–8. IEEE (2023). https://doi.org/10.1109/ETFA54631.2023.10275601
30. Gil, S., Miyazawa, A., Cavalcanti, A., Gomes, C.: Application of FMI 3.0 Synchronous Clocks to Co-simulation of Robotic Systems. In: IEEE/SICE Intl. Conf. on System Integration (SII). IEEE (To Appear 2026)
31. Gil, S., Oakes, B.J., Gomes, C., Frasheri, M., Larsen, P.G.: Toward a systematic reporting framework for digital twins: a cooperative robotics case study. SIMULATION **101**(3), 313–339 (2025). https://doi.org/10.1177/00375497241261406
32. Gil, S., Schou, C., Mikkelsen, P.H., Larsen, P.G.: Integrating Skills into Digital Twins in Cooperative Systems. In: IEEE/SICE Intl. Conf. on System Integration (SII). pp. 1124–1131. IEEE (2024). https://doi.org/10.1109/SII58957.2024.10417610
33. Gomes, C., Kristensen, M.H., Andersen, M.S., Talasila, P., Feng, H., Wright, T., Larsen, P.G.: Digital twin tutorial: The incubator case study. In: Engineering Trustworthy Software Systems: 6th Intl. School, SETSS 2024, Chongqing, China, April 14–21, 2024, Tutorial Lectures. p. 68–101. Springer (2025). https://doi.org/10.1007/978-981-96-4656-2_3
34. Gomes, C., Thule, C., Broman, D., Larsen, P.G., Vangheluwe, H.: Co-simulation: A Survey. ACM Computing Surveys **51**(3), 49:1–49:33 (2018). https://doi.org/10.1145/3179993
35. International Council on Systems Engineering: INCOSE Systems Engineering Vision 2035: Engineering Solutions for a Better World (2021)
36. ISO/IEC 15288:2023: Systems and software engineering. System life cycle processes. International Organization for Standardization, Geneva (2023)
37. ISO/IEC 20924:2024: Internet of things (IoT) and digital twin. International Organization for Standardization, Geneva (2024)
38. ISO/IEC 23247:2021: Automation systems and integration. Digital twin framework for manufacturing. International Organization for Standardization, Geneva (2021)
39. ISO/IEC 30173:2023: Digital twin – Concepts and terminology. International Organization for Standardization, Geneva (2023)

40. Jones, D., Snider, C., Nassehi, A., Yon, J., Hicks, B.: Characterising the Digital Twin: A systematic literature review. CIRP Journal of Manufacturing Science and Technology **29**, 36–52 (March 2020). https://doi.org/10.1016/j.cirpj.2020.02.002
41. Khedr, M.T., Fitzgerald, J.S.: The Composition of Digital Twins for Systems-of-Systems: a Systematic Literature Review (2025). https://doi.org/10.48550/arXiv.2506.20435
42. Kristensen, M.H., Wright, T., Gomes, C., Esterle, L., Larsen, P.G.: DynSRV: Dynamically Updated Properties for Stream Runtime Verification. In: RV 2025 (25th Intl. Conf. on Runtime Verification). Springer (September 2025). https://doi.org/10.1007/978-3-032-05435-7_7
43. Kühne, T.: What is a Model? In: Language Engineering for Model-Driven Software Development. vol. 04101. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI) (2005). https://doi.org/10.4230/DagSemProc.04101.15
44. Kulik, T., Dongol, B., Larsen, P.G., Macedo, H.D., Schneider, S., Tran-Jørgensen, P.W.V., Woodcock, J.: A Survey of Practical Formal Methods for Security. Formal Aspects of Computing (jan 2022). https://doi.org/10.1145/3522582
45. Kulik, T., Gomes, C., Macedo, H.D., Hallerstede, S., Larsen, P.G.: Towards secure digital twins. In: Margaria, T., Steffen, B. (eds.) ISoLA 2022. Lecture Notes in Computer Science, vol. 13704. Springer (2022). https://doi.org/10.1007/978-3-031-19762-8_11
46. Leucker, M., Sánchez, C., Scheffel, T., Schmitz, M., Schramm, A.: TeSSLa: Runtime verification of non-synchronized real-time streams. In: Proc. Annual ACM Symp. on Applied Computing. pp. 1925–1933 (Apr 2018). https://doi.org/10.1145/3167132.3167338
47. Mattila, J., Ala-Laurinaho, R., Autiosalo, J., Salminen, P., Tammi, K.: Using Digital Twin Documents to Control a Smart Factory: Simulation Approach with ROS, Gazebo, and Twinbase. Machines **10**(4) (2022). https://doi.org/10.3390/machines10040225
48. Mazumder, A., Sahed, M., Tasneem, Z., Das, P., Badal, F., Ali, M., Ahamed, M., Abhi, S., Sarker, S., Das, S., Hasan, M., Islam, M., Islam, M.: Towards next generation digital twin in robotics: Trends, scopes, challenges, and future. Heliyon **9**(2), e13359 (2023). https://doi.org/https://doi.org/10.1016/j.heliyon.2023.e13359
49. Medhi, J.: Stochastic processes. New Academic Science Limited (2012)
50. Moyne, J., Qamsane, Y., Balta, E.C., Kovalenko, I., Faris, J., Barton, K., Tilbury, D.M.: A requirements driven digital twin framework: Specification and opportunities. IEEE Access **8**, 107781–107801 (2020). https://doi.org/10.1109/ACCESS.2020.3000437
51. National Academies of Sciences, Engineering, and Medicine: Foundational Research Gaps and Future Directions for Digital Twins. National Academies Press, Washington, DC (2024). https://doi.org/10.17226/26894
52. Oakes, B., Gomes, C., Larsen, P.G., Denil, J., DeAntoni, J., Cambeiro, J., Fitzgerald, J.: Examining Model Qualities and Their Impact on Digital Twins. In: Annual Modelling and Simulation Conference. pp. 220–232. Ontario, Canada (2023)
53. Paulson, L.C.: Isabelle: A Generic Theorem Prover. Springer-Verlag LNCS 828 (1994)
54. Pedersen, M.R., Nalpantidis, L., Andersen, R.S., Schou, C., Bøgh, S., Krüger, V., Madsen, O.: Robot skills for manufacturing: From concept to industrial deployment. Robotics and Computer-Integrated Manufacturing **37**, 282–291 (2016). https://doi.org/10.1016/j.rcim.2015.04.002
55. Platzer, A.: Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics. Springer (2010)

56. Rohmer, E., Singh, S.P.N., Freese, M.: Coppeliasim (formerly v-rep): a versatile and scalable robot simulation framework. In: Proc. of The Intl. Conf. on Intelligent Robots and Systems (IROS). pp. 1321–1326 (2013). https://doi.org/10.1109/IROS.2013.6696520, www.coppeliarobotics.com
57. Saavedra Sueldo, C., Perez Colo, I., De Paula, M., Villar, S.A., Acosta, G.G.: ROS-based architecture for fast digital twin development of smart manufacturing robotized systems. Annals of Operations Research **322**(1), 75–99 (2023). https://doi.org/10.1007/s10479-022-04759-4
58. Siciliano, B., Khatib, O.: Springer Handbook of Robotics. Springer Cham (2016). https://doi.org/10.1007/978-3-319-32552-1
59. Stachowiak, H.: Allgemeine Modelltheorie. Springer-Verlag, Wien and New York (1973)
60. Tang, V., Gomes, C., Lucani, D.E.: Precision on demand: Propositional logic for event-trigger threshold regulation. IEEE Internet of Things Journal **12**(3), 2674–2689 (2025). https://doi.org/10.1109/JIOT.2024.3476922
61. van Amerongen, J.: Dynamical Systems for Creative Technology. Controllab Products B.V., Enschede (2010)
62. Wang, Y., Su, Z., Guo, S., Dai, M., Luan, T.H., Liu, Y.: A survey on digital twins: Architecture, enabling technologies, security and privacy, and future prospects. IEEE Internet of Things Journal pp. 1–1 (2023). https://doi.org/10.1109/JIOT.2023.3263909
63. Woodcock, J., Gomes, C., Macedo, H.D., Larsen, P.G.: Uncertainty quantification and runtime monitoring using environment-aware digital twins. In: Margaria, T., Steffen, B. (eds.) Leveraging Applications of Formal Methods, Verification and Validation: Tools and Trends. pp. 72–87. Springer (2021). https://doi.org/10.1007/978-3-030-83723-5_6
64. Wright, T., Gomes, C., Woodcock, J.: Formally Verified Self-Adaptation of an Incubator Digital Twin. In: Leveraging Applications of Formal Methods, Verification and Validation. Practice: 11th Intl. Symp., ISoLA 2022, Rhodes, Greece, October 22–30, 2022, Part IV. p. 89–109. Springer (2022). https://doi.org/10.1007/978-3-031-19762-8_7