Formal Verification of Digital Twins with TLA and Information Leakage Control

Luwen Huang
Computer Science
University of Texas at Austin
luwen@cs.utexas.edu

Lav Varshney
Electrical & Computer Engineering
University of Illinois at
Urbana-Champaign
varshney@illinois.edu

Karen E. Willcox
Oden Institute for Computational
Engineering & Sciences
University of Texas at Austin

ABSTRACT

Verifying the correctness of a digital twin provides a formal guarantee that the digital twin operates as intended. Digital twin verification is challenging due to the presence of uncertainties in the virtual representation, the physical environment, and the bidirectional flow of information between physical and virtual. A further challenge is that a digital twin of a complex system is composed of distributed components. This paper presents a methodology to specify and verify digital twin behavior, translating uncertain processes into a formally verifiable finite state machine. We use the Temporal Logic of Actions (TLA) to create a specification, an implementation abstraction that defines the properties required for correct system behavior. Our approach includes a novel weakening of formal security properties, allowing controlled information leakage while preserving theoretical guarantees. We demonstrate this approach on a digital twin of an unmanned aerial vehicle, verifying synchronization of physical-to-virtual and virtual-to-digital data flows to detect unintended misalignments.

CCS CONCEPTS

• **Computer systems organization** → **Embedded and cyber-physical systems**; **Real-time system specification**; • **Hardware** → **Safety critical systems**; • **Theory of computation** → Formal languages and automata theory.

KEYWORDS

Digital twins, cyber-physical systems, safety-critical systems, TLA, formal design and specification, formal verification, model checking
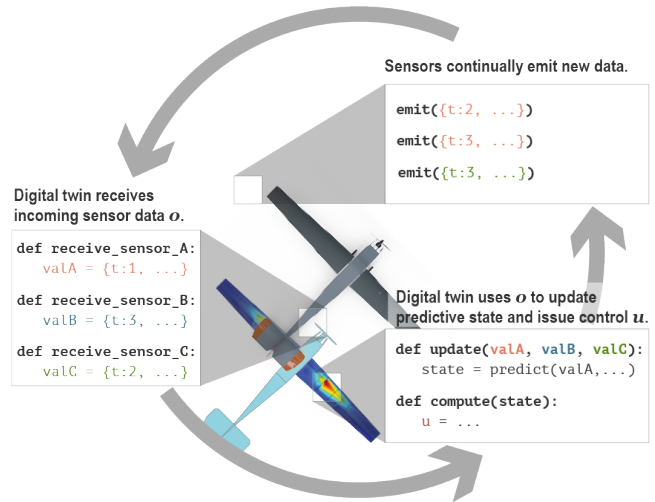
Figure 1: Digital twin consisting of a physical system (unmanned aerial vehicle), a virtual representation (structural health models), and bidirectional connections among components.

1 INTRODUCTION

This paper describes a formal methodology to design and model a digital twin and prove its correctness properties using the Temporal Logic of Actions (TLA). We employ the National Academies' definition: "A digital twin is a set of virtual information constructs that mimics the structure, context, and behavior of a natural, engineered, or social system (or system-of-systems), is dynamically updated with data from its physical twin, has a predictive capability, and informs decisions that realize value. The bidirectional interaction between the virtual and the physical is central to the digital twin" [1].

An example digital twin is illustrated in Fig. 1. In this scenario, an unmanned aerial vehicle (UAV) flies a mission while transmitting sensor data to a digital twin. The digital twin, designed to mirror the UAV's structure, context and behavior, processes the incoming sensor data, o, maintains a predictive model of the UAV's structural health, s, and generates control execution commands, u. Even when individual components, like the command-generation function, operate correctly, the system can still fail due to orchestration issues. For instance, consider the sensor observations valA, valB, and valC, arriving concurrently but with different timestamps t=1, t=3, and t=2, respectively. These readings, emitted at different intervals and

with transmission delays, create orchestration challenges. To ensure the correct state, the update function must incorporate consistent sensor inputs, and the UAV must process incoming commands reliably to follow its intended path. These examples highlight that verifying individual component correctness is not enough; ensuring the digital twin's overall orchestration is equally important.

Various formal and technological approaches address aspects of correctness in cyber-physical systems. For instance, verifying control and timing is well-researched (see Sec. 2), but control verification alone does not ensure system-level orchestration. Technological solutions, such as RabbitMQ for asynchronous data handling, address only specific areas of digital twin functionality. Moreover, simply adding technological components does not offer formal guarantees, often a crucial need in safety-critical environments where digital twins may be deployed. This paper introduces a novel methodology for formally reasoning about digital twins at the level of system orchestration. We introduce the following innovations:

(1) **Formal system specification**: A new method to construct formal, high-level specifications of digital twins using TLA. Our approach derives a finite state machine model from the digital twin probabilistic graphical model (PGM) [2], giving a mathematically rigorous way to specify digital twins in general.
(2) **Model augmentation**: A novel augmentation of the digital twin PGM framework to model distributed communication and the corresponding state machine translation.
(3) **Abstraction methodology**: A set of principled guidelines for abstracting the physical and computational complexities of digital twins into state transition actions.
(4) **Weakening of formal properties**: A novel approach to relax formal security properties, such as non-interference, by bounding the utility of revealed information within digital twin bidirectional flows, thereby limiting impact on system identification rather than relying on generic information-theoretic bounds.

The remainder of this paper is organized as follows. Sec. 2 places our approach in the existing literature. Sec. 3 details the state machine derivation. Sec. 4 demonstrates a practical application by constructing and verifying a UAV digital twin, with relaxed security properties that provide formal bounds on information leakage between the physical and digital components. Finally, Sec. 5 presents the results of our verification efforts on the UAV digital twin.

2 RELATED WORK

Our research contributes to the field of cyber-physical systems, with particular focus on the expanding concept of digital twins. As digital twin technology continues to evolve rapidly, it is important to delineate how our approach both aligns with and diverges from existing work.

Digital Twin frameworks. Various works have suggested digital twin design approaches that range from informal, flow chart-based design [3], [4] to technology-specific solutions [5, 6]. Unlike these approaches, our methodology is technology-neutral. Our approach offers a generalizable abstraction for digital twin design that is grounded in mathematically rigorous formal verification principles.

Verification of cyber-physical systems. Verification of cyber-physical systems is a dynamic and expansive area of research [7], with much work in safe autonomy and control [8–16]. There is also considerable discourse on the challenges of distributed cyber-physical systems [17–19]. For example, [20] extends the Lingua Franca coordination language to handle network failures, [21, 22] discuss the need for timing considerations in distributed environments, and [23–25] offer methods to achieve deterministic timing in control executions. In contrast, our paper provides a methodology for orchestrating digital twins at the system level. Aligning with the NASEM definition of a digital twin, our approach emphasizes the critical importance of bidirectional interactions and orchestration, offering a broader, systemic perspective that diverges from the control-centric emphasis found in much existing literature.

Distributed systems. The application of TLA in distributed computing systems is well-documented [26–29], with notable applications including its use at Amazon Web Services for managing distributed resources [30]. While TLA has proven effective in addressing the complexities of distributed computing, the specification of digital twins presents unique challenges that extend beyond traditional distributed systems: First, digital twins require the consideration of diverse hardware components, which goes beyond the typical software and network considerations found in distributed systems [17]. Second, digital twins often incorporate predictive models that provide probabilistic outputs and may adapt dynamically based on real-time data. Third, digital twins necessitate continual, real-time bidirectional exchanges to maintain synchronization between the physical and digital entities. Our research applies TLA to address these aspects, offering a formal, verifiable system perspective for digital twins. To our knowledge, this represents a novel application of TLA in the context of digital twins. Moreover, we provide a systems-theoretic approach to give formal statistical guarantees on information leakage during communication between the physical system and its digital twin.

3 DIGITAL TWIN AS A STATE MACHINE

Our first result is formalizing a digital twin as a state machine, rigorously derived from the digital twin Probabilistic Graphical Model (PGM) framework proposed in [2] and since adopted to describe digital twins in a variety of applications [31, 32]. Appendix A provides a background of the digital twin PGM framework. Our formalization uses TLA to describe the digital twin as a finite state machine (FSM). Indeed, Markov models as in PGMs are a stochastic version of FSMs. For background on TLA, see [33–35]. Throughout this section, we use examples from our application instance of a UAV digital twin. However, we emphasize and show that our methodology is broadly applicable.

3.1 State Machine Derivation

Here we detail our novel derivation of a state machine representation from the digital twin PGM. We specify the digital twin as a state machine that transitions from one state to the next, governed by transition logic:

$$\text{Digital Twin} := \mathcal{I} \wedge \mathcal{N} \wedge \mathcal{F} \tag{1}$$
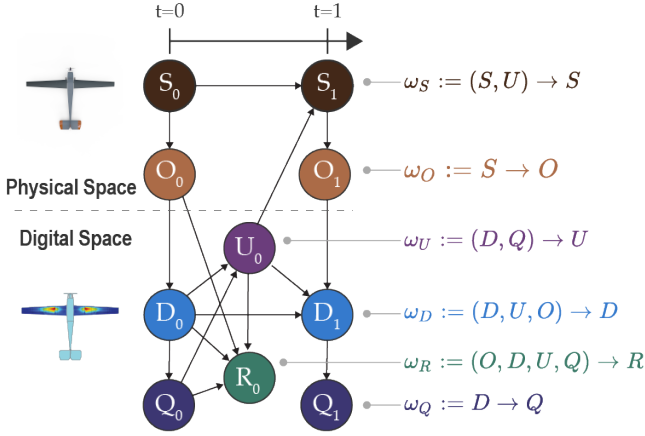
Figure 2: Derivation of state machine processes from PGM representation. Nodes represent variables and edges between nodes represent the dependence of the destination node on the parent node. The subscript X_t denote the variable X's state at time t.

Figure 3: PGM with distributed communication required for two processes: (1) $O \rightarrow D$ and (2) $U \rightarrow S$.

Here, (1) states the state machine of the digital twin is defined by a conjunction (AND) of an initial state predicate $\mathcal{I}$, a next state predicate $\mathcal{N}$, and a set of fairness conditions $\mathcal{F}$. The initial state predicate specifies the valid starting conditions, the next state predicate outlines the permissible transitions that variables can undergo, and the fairness conditions provide assumptions about how transitions are executed. Specifically, the next state predicate:

$$\mathcal{N} := \omega_1 \vee \cdots \vee \omega_N \vee \mathcal{T} \qquad (2)$$

employs the logical disjunction (OR) to indicate that at any given step in the state machine, one out of N possible processes ω_n can occur, or the system can reach a termination condition $\mathcal{T}$. By allowing only one process to execute at a time, we model concurrency by considering the possible orderings, or interleavings, of process execution, abstracting away timing specifics.

To model digital twin orchestration, we identify the specific operations or "processes" through which variables within the system alter their states. These processes are dictated by the relationships encoded within the DT PGM. Each variable v_i in the model transitions based on the states of other variables that directly influence it — the variable node's parents in the graphical model. Formally, the set of processes Ω is defined as:

$$\Omega = \{\text{Define } \omega := W(v_i) \rightarrow v_i \mid v_i \in V \wedge W(v_i) \neq \varnothing\} \qquad (3)$$

Here, $W(v_i)$ is the set comprising the parents of v_i, and the transition function $\rightarrow$ denotes the computation that updates v_i based on these influences. This definition preserves the system dependencies by defining that each variable's change is a direct result of its process' inputs.

Fig. 2 shows the PGM describing our example UAV digital twin with six variables: (1) physical state S which represents the structural health of the UAV; (2) Observational data O representing sensor data; (3) Digital state D, which represents the digital twin's estimate of the UAV's structural health; (4) control U representing
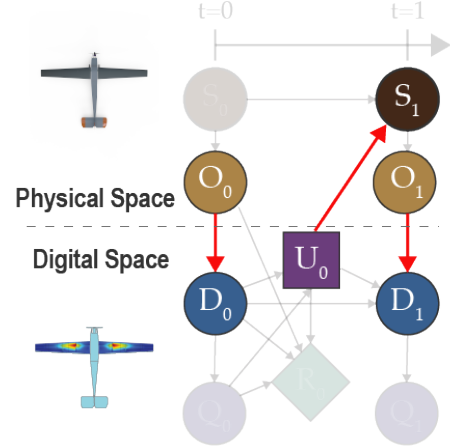
the computed control; (5) Quantity of interest Q, which represents quantities of interest computed by the digital twin; and (6) Reward R, representing metrics for success as dependent on O, D, U and Q. Applying (3) to the PGM yields the set of processes $\Omega = \omega_S \vee \omega_O \vee \omega_U \vee \omega_D \vee \omega_R \vee \omega_Q$ where $\omega_S := (S, U) \rightarrow S$, $\omega_O := S \rightarrow O$, $\omega_U := (D, Q) \rightarrow U$, $\omega_D := (D, U, O) \rightarrow D$, $\omega_R := (O, D, U, Q) \rightarrow R$ and $\omega_Q := D \rightarrow Q$. Fig. 2 illustrates the mapping of PGM encodings to state machine processes.

3.2 Modeling Distributed Communication

The second major contribution of our work is the novel augmentation of the digital twin PGM to account for the challenges of distributed comopnents, and the corresponding translation into the state machine representation.

The graphical model in Fig. 2 assumes that variable values are read deterministically. This is often not the case in digital twins where components are distributed and rely on message passing to communicate with each other. With distributed components, there is additional *uncertainty* in the input values that are actually used by a process, stemming from issues such as network reliability and traffic. For instance, as illustrated in Fig. 3, the process $O \rightarrow D$ requires the value of O, which is transmitted via distributed messaging — in this case, a wireless network channel. The perturbation of the distributed messaging in a PGM might even be adversarial and so a worst-case analysis may be needed [36].

Our novel augmentation of the PGM constructs a new variable N to represent the uncertainty of the messaging channel and a new variable X_{in} for every variable X whose value is communicated over the messaging channel. These noise and channel output variables are just like in information-theoretic models of communication [37], but considering semantics of logic [38]. First, identify the set of variables $\mathcal{X}$ whose value is communicated over distributed messaging, i.e. $\mathcal{X} = \{X_i \mid X_i \in W(v_i) \wedge v_i \in V \wedge X_i \rightarrow v_i \text{ is distributed}\}$. For every $X_i \in \mathcal{X}$, we create an intermediary variable $X_{i,in}$ and a network variable N_{X_i} to represent the value of X_i actually received. We reconfigure the incident edges of X_i such that new edges point

---

Algorithm 1 Augment PGM for communication uncertainty

Input: PGM $\mathcal{G}$
Output: Augmented PGM $\mathcal{G}'$
   Set $V \leftarrow$ set of nodes in $\mathcal{G}$
   Set $E \leftarrow$ set of edges in $\mathcal{G}$
   for v in V **do**
      Define $v \rightarrow w$ to be the outgoing edge from v to node w
      if $v \rightarrow w$ is distributed **then**
         Create v_{in} as new node, $V \leftarrow V \cup v_{\text{in}}$
         Create n_v as new node, $V \leftarrow V \cup n_v$
         Remove edge $v \rightarrow w$ from E
         Create new edge $v \rightarrow v_{\text{in}}$, $E = E \cup v \rightarrow v_{\text{in}}$
         Set new edge $n_v \rightarrow v_{\text{in}}$, $E \cup n_v \rightarrow v_{\text{in}}$
         Set new edge $v_{\text{in}} \rightarrow w$, $E \cup v_{\text{in}} \rightarrow w$
      end if
   end for
   return V, E

---

from X_i to $X_{i_{\text{in}}}$, N_{X_i} to $X_{i_{\text{in}}}$, O to N_{x_i}, and $X_{i_{\text{in}}}$ to V_i. Algorithm 1 details the augmentation algorithm.

For example, Fig. 4 shows a subgraph of the resulting augmentation applied to variable O. The augmentation introduces three new processes: (1) ω_{N_O}, which represents the optional dependence of the messaging channel on the value of O. For instance, some messaging channels may be susceptible to large data payloads and may degrade as traffic increases. (2) $\omega_{O_{\text{in}}}$, which represents the dependency of O_{in} on both the message that was sent and the state of the network. (3) $\omega_{O_{\text{in}}}$, which replaces the original process $O \rightarrow D$ to model the fact that the process input is the received variable O_{in}, instead of sent variable O. Our state machine formalization further elaborates on fairness, termination, and complexity abstraction, detailed in Appendix B.

4 SPECIFICATION OF UAV DIGITAL TWIN

This section applies our proposed methodology to the design, specification, and verification of a UAV digital twin.

4.1 The UAV and its Digital Twin

The physical counterpart of this digital twin is a custom-built, fixed-wing UAV equipped with advanced wireless sensors and power hardware, with construction described in [39] and shown in Fig. 5.
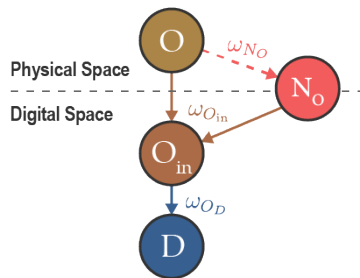
Figure 4: Augmentation for O, which is communicated over a distributed channel to D.
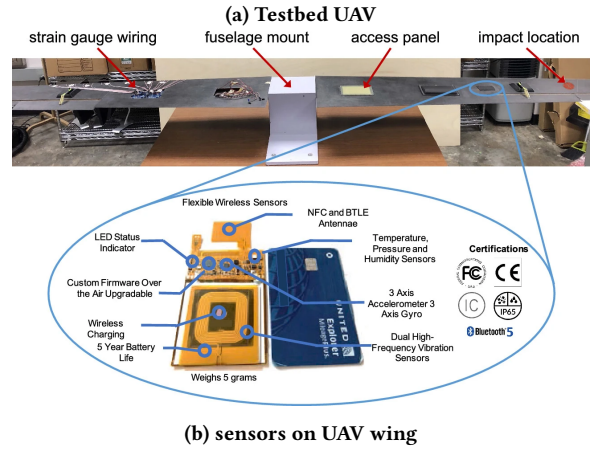
Figure 5: Reproduced with permission from [39]: testbed UAV (top) equipped with individually-transmitting Bluetooth sensors (bottom)

The sensors, attached to the UAV's wings as in Fig. 5b, measure observational data such as temperature and strain in real-time during flight. The UAV also features an onboard computer to process incoming control commands from the digital twin, where commands are executed as maneuvers. Given the potential unreliability of the communication channel, a primary design challenge is ensuring that delayed control messages are processed accurately to maintain the UAV's operational integrity.

A digital twin of this UAV would continually process incoming observational data to generate and transmit control commands tailored for the UAV. The digital twin would also maintain a dynamic predictive model of the UAV's structural health, ensuring synchronization with the UAV's actual physical state. This synchronization is achieved through real-time computations that integrate new observational data into the ongoing assessment of the UAV's condition. A key design challenge of the digital twin is its ability to accurately reflect the UAV's physical state despite potential latency issues and concurrent, incoming data streams.

Our implementation builds upon the digital twin in [2], implemented as a collection of Robot Operating System (ROS2) Python modules. However, the original implementation primarily served as a proof-of-concept for the PGM framework and did not address several real-world challenges such as handling concurrent incoming observational messages and ensuring reliability over unstable
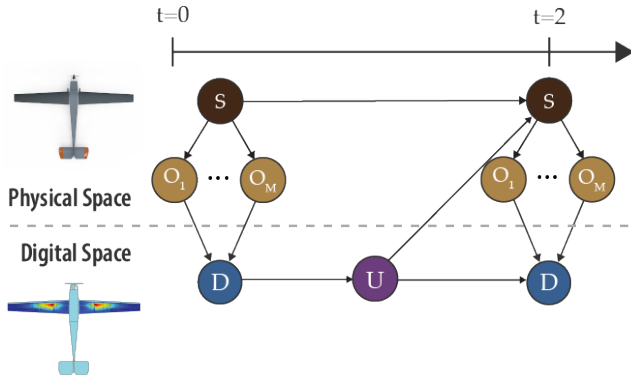
Figure 6: PGM for UAV Digital Twin

communication channels. Our objective is to construct a design to manage these complexities and achieve reliable orchestration under realistic operational conditions.

4.2 The UAV State Machine

We apply our augmentation methodology from Algorithm 1 to construct an augmented PGM that accounts for the distributed messaging channels in the system. Because each sensor transmits independently, we specify each sensor's connection as separate variables, $N_1 \ldots N_m$. We also define a separate variable for the transmission of control commands, N_u. These definitions let us reason about bidirectional flows individually. In addition to new variables for each data transmission path, the augmentation also introduces new nodes for received observational data $O_{1,\text{in}} \ldots O_{m,\text{in}}$ and received control command U_{in}. The augmented PGM is depicted in Fig. 7.
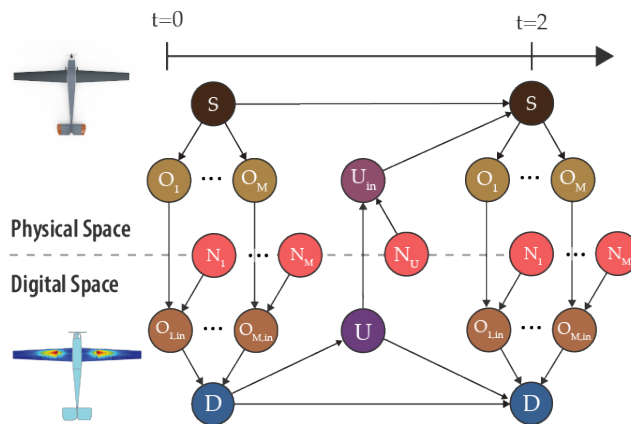

Figure 7: Augmented PGM modeling distributed communication

4.3 System Abstraction

This subsection applies our abstraction methodology to model concrete state transitions within the UAV digital twin's state machine. In abstracting complex system dynamics into simpler, formal state transitions, our goal is to balance fidelity with tractability: while a more granular formalization more accurately reflects real-world dynamics, it becomes less scalable in terms of formalization effort and verification time. We organize this section by systematically addressing each variable involved in the UAV system. For each variable, we first describe its real-world characteristics and then its corresponding abstraction. Following this, we delineate how each variable evolves in real life and how we formulate its state transition within the state machine.

Physical state (S) The physical state S of the UAV represents its structural health, which is influenced by the stresses of executed maneuvers. It is not possible for the digital twin to know the ground truth of S at runtime; instead, it must be inferred through sensor data. The UAV's structural integrity is subject to degradation, quantified by δ damage, which occurs with a non-zero probability dependent on the executed control. This probabilistic damage is governed by the dynamics shown in (4) in Table 1.

In our abstraction, we model the structural health S as a discrete variable ranging from 0 (total structural failure) to 100 (perfect health). Our model simplifies probabilistic damage to a nondeterministic state transition where the structural state either remains unchanged or is reduced by $\delta = 1$ damage, shown in (5) in Table 1. Finally, because damage occurs concurrently with control execution, both actions are modeled as a single atomic operation in (5) in Table 1, where the value of the next executed control u_e is assigned the control command u_{in}.

Table 1: Transition $(S, U) \rightarrow S$: Evolution of physical state

Real-world process	Abstraction
$\phi = \begin{cases} 0.05 & \text{if } u = 3 \\ 0.01 & \text{if } u = 2 \end{cases}$ (4)	$\begin{aligned} & \wedge u_e' = u_{\text{in}} \\ & \wedge \vee s' = s \quad\quad (5) \\ & \quad\vee s' = s - \delta \end{aligned}$

Observational data (O) The observational data, denoted as $O = O_1 \ldots O_M$, are noisy, timestamped sensor measurements of the UAV's structural health, taken by M sensors, indexed as $m = 1 \ldots M$. Sensor measurements inherently vary slightly from the actual structural health and each other due to sensor precision and environmental interference. Empirical data (sample shown in (6)), show typical small deviations from the ground truth value S.

In our abstraction, each sensor measurement O_m is represented as the UAV's structural health value perturbed by some nondeterministic noise $\epsilon \in \{-1, 0, 1\}$.

Distributed messaging of observational data (N, $O_{m,\text{in}}$) In the UAV digital twin, the communication of observational data via Bluetooth introduces complexities due to the potential unreliability of the wireless channels.
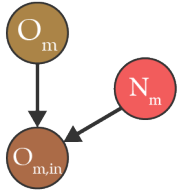
In our abstraction, we construct separate variables for each Bluetooth channel ($N_1 \ldots N_M$) and for the received data $O_{1,\text{in}} \ldots O_{M,\text{in}}$.

Because our concern is at a higher level than the details of sensor and transmission operations, we treat the processes of data

Table 2: Transition $S \to O_m$: Generate observational data

Real-world process	Abstraction
$o_m - h = [0.57, 0.66,$ $0.31, -0.12,$ $0.42, -0.93,$ $0.61, \ldots]$ (6)	$o'_m = s + \epsilon$ (7)

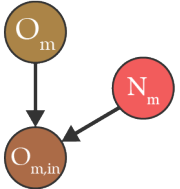**Table 3: Transition $O_m \to N_m$: Transmit observational data**

Real-world process	Abstraction
	$n'_m = \text{push } n_m, o_m$ (8)

generation, abstracted in (7), and transmission, abstracted in (8), as mutually-atomic. This abstracts the generation and immediate transmission of data as a single, indivisible operation, as in (9):

$$\text{ObserveEmitObsAtomic} := (O_m \to N_m) \land (S \to O) \quad (9)$$

The received value of a sensor message O_m is represented by variable $O_{m,\text{in}}$. Per our methodology in Appendix B.1, to model the unreliable receiving of messages, we remove the element at randomly-chosen index i from queue n_m, and we add it to the received messages collection $o_{m,\text{in}}$. We impose the strong fairness condition that the correct message ($i = 1$) is always eventually delivered.

Table 4: Transition $(O_m, N_m) \to O_{m,\text{in}}$: Receive observational data

Real-world process	Abstraction
	Let $i \in [1, \eta]$ $\land$ Remove $n_m[i]$ $\land$ $o'_{m,\text{in}} = \text{push } o_{m,\text{in}}, n_m[i]$ $\land$ $SF(i = 1)$ where $1 \le \eta \le \text{Length } n_m$

Digital state (D) The digital state D represents the estimated structural health of the UAV, modeled as a variable within the range $\{1 \ldots 100\}$. This estimation is computed by a black-box model ψ, which outputs a predictive distribution for D. While the internal computations of each model remain undisclosed, output characteristics are discovered through prior statistical analysis.

Our abstraction retains the dependency of D on previous state D^{t-1}, last control computed U^{t-1} and the latest observational data $\mathbf{O}^t_{\text{in}}$. To enhance the model's tractability, we use known characteristics of ϕ to constrain the number of possible states for D. For instance, when analyzing the conditional probability for D_O, shown in (10), where D varies with $\mathbf{O}^t_{\text{in}}$ while keeping other factors constant, we observe that non-positive sensor observations significantly widen the range of possible values for D. Otherwise, D typically fluctuates within a normal distribution $\mathcal{N}(d, \sigma^2)$, where the variance σ^2 is influenced by the type of control executed. To keep the abstraction tractable and focused on the most critical scenarios, we constrain D to fluctuate within two standard deviations of the mean. This constraint is reflected in our abstraction, where $\zeta_2 \in [-1, 1]$ and $\zeta_3 \in [-5, 5]$ are set to represent the two standard deviation bounds for controls $u = 2$ and $u = 3$, respectively, rounded to the nearest integers.

Table 5: Transition $(D, U, O_{m,\text{in}}) \to D$: Update digital state

Real-world process	Abstraction
$D \sim \psi(D^{t-1}, U^{t-1}, \mathbf{O}^t_{\text{in}})$ $D_O \sim \begin{cases} \text{U}(0, d) & \exists o_i \le 0 \\ \mathcal{N}(d, \sigma^2) & \text{otherwise} \end{cases}$ (10)	IF $\exists o_{m,\text{in}} : o_{m,\text{in}} \le 0$ $\quad d' = 0 \ldots d$ ELSE $\quad$ IF $u = 2$ (11) $\quad\quad d' = d + \zeta_2$ $\quad$ ELSE $\quad\quad d' = d + \zeta_3$

Control (U) The control U is a command that instructs the UAV to execute either a $3g$ or $2g$ turn. The control is computed via a optimization model.

In our abstraction (Table 6), the control U is simplified to decision-making criteria based primarily on the UAV's estimated structural health D. This simplification is grounded in a prior analysis of the optimization model's outputs [2], which reveal that the value of D primarily dictates whether the control U can be set to 3.
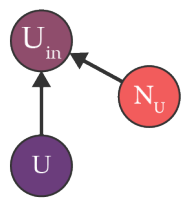
Table 6: Transition $D \to U$: Compute and transmit control

Real-world process	Abstraction
$u \in \begin{cases} \{3\} & d \ge D_{\min} \\ \{2, 3\} & \text{otherwise} \end{cases}$ (12)	IF $d \ge D_{\min}$ $\quad \lor u' = 3$ $\quad \lor u' = 2$ ELSE $\quad u' = 2$

Distributed messaging of control (N_U, U_{in}) Handling control messages is managed similarly to transmission and reception of observational data. In our abstraction (Table 7), we assume that the correct message will eventually be delivered, and we treat the processes of computing a control decision and transmitting a control as mutually atomic operations, combined into a single, indivisible process to reduce complexity.

Termination. Our UAV example uses specific termination conditions to reflect real mission parameters. Termination occurs when: (1) UAV reaches the maximum number of executed maneuvers $C_{\max}$;

Table 7: Transition $(U, N_u) \rightarrow U_{\mathbf{in}}$**: Receive control**

Real-world process	Abstraction
	Let $i \in [1, \eta]$ $\wedge$ Remove $n_u[i]$ $\wedge$ IF $n_u[i].t > u_{\mathrm{in}}.t$ $\qquad u'_{\mathrm{in}} = n_u[i]$ $\qquad$ ELSE $\qquad u'_{\mathrm{in}} = u_{\mathrm{in}}$ $\wedge SF(i = 1)$ where $1 \le \eta \le$ Length n_u

(2) digital twin exceeds a predefined maximum runtime $T_{\max}$; or (3) digital twin estimates the UAV's structural health as non-positive, and all sensor readings concurrently indicate non-positive values, suggesting critical system failure.

4.4 Specifying Properties

The core property of interest in the UAV digital twin is *synchronization*—the continuous, bidirectional feedback loop ensuring that the physical and digital entities reflect each other accurately. We define our primary synchronization property as:

P_1: *The physical and digital twins must be eventually synchronized.*

We use the term "eventually" to describe that synchronization will always be achieved, without binding it to a specific timeframe. To detail what synchronization entails, we deconstruct this overarching property into more granular sub-properties, guided by methodological questioning — *how*, *what*, and *why* [40, 41] — with engineers and stakeholders. We discuss in more detail how we specify properties in Appendix B.2. The resulting property-part diagram, depicted in Fig. 8, illustrates a subset of these properties.
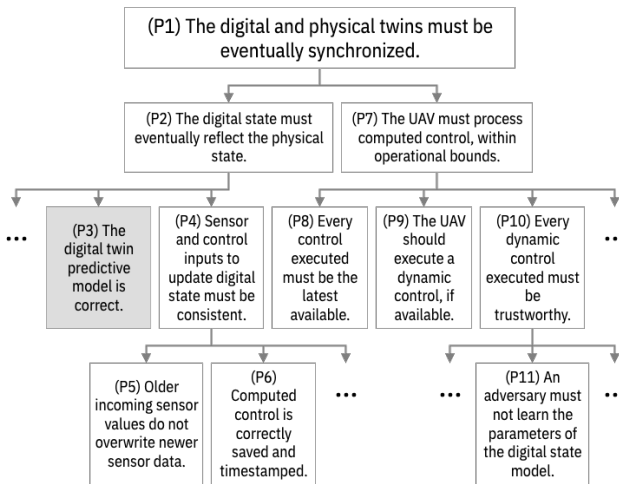

Figure 8: Partial property-part diagram showing a subset of properties

4.5 Weakening Formal Verification with Statistical Guarantees

Synchronization correctness require certain security properties to be satisfied. For example, in Fig. 8, P_{11} requires that an adversary cannot infer information about the digital state model, which is necessary for the trustworthiness of messages exchanged between the physical and digital twins. This property falls under a class of security guarantees known as non-interference, a standard approach for formalizing information flow within a system. A process r_1 is non-interfering with another process r_2 across system $\mathcal{M}$ if r_1's input to $\mathcal{M}$ has no effect on M's output to r_2 [42]. Different variations of noninterference exist [43], including generalized non-interference (GNI) which extends noninterference to probabilistic systems by mandating that for every pair of traces b and b', there exists a third trace b'' such that b'' agrees with the low-security inputs and b'' agrees with the high-security outputs [44]. The practicality of noninterference is well-known to be problematic [45], and as of state-of-the-art, obeying GNI is still an impractical constraint on digital twin systems. Here, we introduce a novel weakening of GNI with respect to particular secret digital twin parameters, where we allow some information leakage while still maintaining formal bounds on the amount of relevant information leaked.

Notably, we measure information leakage through a *system identification* perspective [46] rather than a generic information-theoretic view [45], considering what systems-theoretic understanding of the digital twin is leaked rather than just the number of bits about it, which may or may not be relevant to adversarial action. This is different from [47] which looks at state estimation rather than system identification, and [48], which is also quite different.

For example, consider the content of the communication involving the current health of the physical counterpart and the next action it is going to take. The change in health depends on the action taken and some system randomness. More concretely, let $h(t) \in \mathbb{N} \cup \{0\}$ denote the health of the system at time t. The system can take m possible actions, indexed by $\{1, 2, \ldots, m\}$. Let $a(t) \in \{1, 2, \ldots, m\}$ denote the action the system takes at time t. We assume the change in health $h(t) - h(t + 1)$ is a Poisson random number drawn with rate $\lambda_{a(t)}$, independent of all other changes in health:

$$h(t) - h(t + 1) = -\Delta h(t + 1) \sim \mathrm{Poisson}(\lambda_{a(t)}).$$

This essentially implies that the health model of the system is given by $(\lambda_1, \lambda_2, \ldots, \lambda_m)$.

An adversary intercepts the communication between the digital twin and the physical counterpart, and knows the values of $h(t)$ and $a(t)$. We want to determine whether the system's health model is compromised by this information leakage. So the estimation problem here is that given $\{(h(\tau), a(\tau)) : 1 \le \tau \le t\}$, we want to figure out $\lambda_1, \lambda_2, \ldots, \lambda_m$.

In the general case, let us assume that λ_i and λ_j have no relation to each other (this may not be very practical since we often know which actions are costlier than others, but let us nevertheless make this simplifying assumption). So for estimating each λ_i, we only consider the set of times $\{\tau : a(\tau) = i\}$. In the absence of any prior,

a good estimator for this would be

$$\hat{\lambda}_i = -\frac{1}{|\{\tau : a(\tau) = i\}|} \sum_{t \in \{\tau : a(\tau) = i\}} \Delta h(t+1). \qquad (13)$$

Using standard probability results, this estimator has the following properties.

THEOREM 1. *The estimator in* (13) *satisfies*

(i) $\mathbb{E}\left[\hat{\lambda}_i\right] = \lambda_i$.

(ii) $\mathbb{P}\left(|\hat{\lambda}_i - \lambda_i| \geq \epsilon\right) \leq \frac{\lambda_i}{N_i \epsilon^2}$, *where* $N_i = |\{\tau : a(\tau) = i\}|$.

Thus as the number of times a particular action is taken increases, we get a more accurate estimate of the hit to health from that action and so we directly get a statistical guarantee on information leakage about the system properties. In general, finite-sample bounds from system identification theory [49–51] can characterize such digital twin-relevant information leakage. With this weakening approach, we are able to satisfy property P_{11}, which would otherwise fail with a purely model-checking approach.

5 EVALUATION

Our baseline specification for the UAV digital twin encompasses various parameters: $M = 2$ sensors, each with a maximum message delay of $\eta = 2$, a total of $C_{\max} = 3$ possible mission maneuvers, and a maximum system runtime of $T_{\max} = 4$. This specification manifests as 15 distinct processes and 18 variables, including auxiliary variables for supporting property verification, with 25 properties covering core system behavior. The TLA code closely mirrors the abstraction models presented in Sec. 4. An example code listing is shown in Appendix C.

5.1 Model Checking the State Space

The state space generated by the UAV digital twin's specification is combinatorially large, as each distinct process introduces a different potential interleaving, with every variable within these interleavings capable of assuming various values. We visualize this state space as a directed acyclic graph (DAG) in Fig. 9, where each vertex represents a unique state—specific values assigned to variables—and edges depict transitions between these states. This graph is inherently a DAG, as it includes a model-checked guarantee of termination. In Fig. 9, terminating states are highlighted in orange, while ongoing states are in black. The graph's initial state, depicted as a blue vertex (1), bifurcates into two principal pathways: the physical twin's processes (2) and the digital twin's processes (3). To highlight one possible pathway: from state (2), the system progresses to state (4) and then to (7), culminating in state (15). This final state indicates termination triggered by the UAV achieving the prescribed number of maneuvers.

Model checking is resource-intensive due to the vast size of the state space. On a hardware setup with 10 cores and 16 GB of RAM allocated to the TLC model checker, completing a single baseline model checking session requires approximately 15 hours. To evaluate scalability, we vary model parameters individually while keeping others constant. Increasing the number of sensors or the permissible message delay notably expands the state space by introducing more potential message interleavings. For example, with

Table 8: Model parameters impact state space complexity

Specification	Distinct States	Total States
Baseline	12 551 574	33 960 246
+1 health ($S = 3$)	24 668 110	66 833 826
+1 sensor ($M = 3$)	13 534 045	41 966 573
+1 delay ($\eta = 3$)	15 307 358	50 720 696
±1 noise ($\epsilon = \pm 2$)	15 804 834	42 619 510
+1 process ($|\Omega| + 1$)	1 227 202	3 231 322

two sensors and a maximum message delay of $\eta = 2$, the model generates 13 534 045 states. Expanding to three sensors increases the state space to 41 966 573, requiring two days to check on our hardware. We also examine the impact of atomicity assumptions by modifying the process $(S, U) \rightarrow S$, which asserts that the execution of control and the incurring of damage occur atomically (see Table 1). By splitting the process into two interleaved, non-atomic steps, we unexpectedly observe a significant reduction in the state space — from 12 million to just one million distinct states. We hypothesize that this decrease results from the model checker simplifying invariants and pruning redundant states more effectively. This finding indicates that atomic assumptions do not always lead to larger state spaces and, in some cases, may simplify specification design. Table 8 summarizes the impact of varying model parameters.

5.2 Safety and Liveness Violations

Throughout the development of our specification, we used an iterative approach that, while refining the design, also continuously exposed gaps that led to property violations. For instance, during a model checking session, we encountered a violation of property P_8: *The executed command must be the latest command seen thus far*, formalized as $\Box[u_{\text{executed}} \neq \varnothing \wedge u'_{\text{executed}} \neq \varnothing \implies u'_{\text{executed}}.t > u_{\text{executed}}.t]$. The sequence of state transitions leading to this violation, simplified for clarity, includes the following key steps:

(1) **Initial State**: The system begins in its initial configuration.
(2) **Execute Command**: The UAV executes a backup command (timestamped $t = 1$) because no dynamic command is available.

```
u_executed = {t: 1, name: "Backup", type: 2 }
```

(3) **Compute and Emit Command**: The digital twin computes and emits a dynamic command (timestamped $t = 1$).

```
u = {t: 1, name: "Dynamic", type: 3 }
n_u = [{t: 1, name: "Dynamic", type: 3}]
```

(4) **Receive Command**: The UAV receives this latest computed dynamic command.

```
u_in = {t: 1, name: "Dynamic", type: 3 }
n_u = []
```

(5) **Execute Command**: The UAV executes the received dynamic command, violating P_8, as the command (timestamped $t = 1$) is stale and should not have been executed.

```
u_executed = {t: 1, name: "Dynamic", type: 3 }
```

The progression of states leading to this violation is depicted in Fig. 10, where state (15) represents the state where the violation
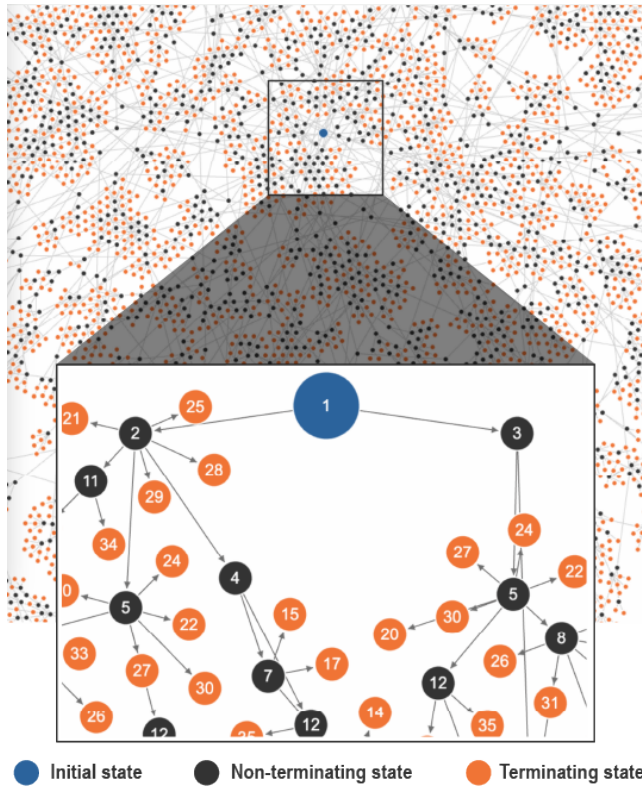
Figure 9: Visualization of state space: nodes represents states and edges represent transitions from state to state

occurs. This issue stems from a critical oversight in the **Receive Command** process, where we had failed to implement a timestamp validation check for incoming command messages before their acceptance into the u_{in} variable. While this oversight might seem straightforward to address in hindsight, it was easily overlooked during the initial stages of specification development. Fig. 11 shows the specification pre- and post-fix. This example underscores the importance of our iterative specification and model checking approach, particularly as design complexity increases, where seemingly fixes fixes can become obscured and go unnoticed.

6 DISCUSSION

This paper presents a methodology for developing formally verifiable DT designs using TLA by transforming the PGM framework into a finite state machine with an augmentation for distributed communication. This approach enables the abstraction of complex distributed DT dynamics, allowing for the verification of synchronization properties. Because traditional formal methods have limitations, particularly with strict security definitions, we address this with a novel weakening method that combines formal verification with statistical guarantees. This allows controlled information leakage while ensuring these weakened properties align with the property-part diagram used in model checking. Despite the challenge of state space explosion, a common issue in model checking [52], even models with small parameters revealed early
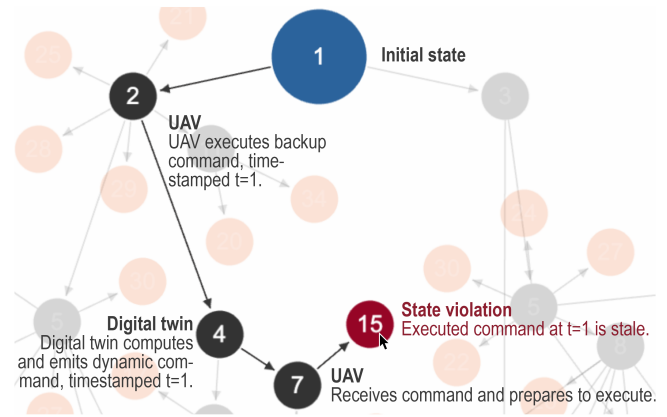

Figure 10: Graph visualization showing the path that leads to safety property violation

```
PT_ReceiveControlDelayed(m_idx) ==
    /\ IF (s > 0 /\ u_executed_count <= MaxManeuvers)
       THEN /\ u_in' = n_u[m_idx]
            /\ n_u' = Remove(n_u, m_idx)
```

(a) Previously: Transition action for UAV receiving commands violates property P_8

```
PT_ReceiveControlDelayed(m_idx) ==
    /\ IF (s > 0 /\ u_executed_count <= MaxManeuvers)
       THEN /\ IF n_u[m_idx]["t"] > u_in["t"]
               THEN /\ u_in' = n_u[m_idx]
                    /\ n_u' = Remove(n_u, m_idx)
```

(b) After: Transition action for UAV receiving commands with addition of a timestamp check

Figure 11: Example of a property violation and subsequent fix

design errors. This iterative process highlights the value of formal verification in safety-critical systems, and future work will focus on bridging the gap between high-level formal specifications and practical digital twin implementations.

ACKNOWLEDGMENTS

REFERENCES

[1] National Academies of Sciences, Engineering, and Medicine. *Foundational Research Gaps and Future Directions for Digital Twins*. The National Academies Press, Washington, DC, 2024.
[2] Michael G. Kapteyn, Jacob V.R. Pretorius, and Karen E. Willcox. A probabilistic graphical model foundation for enabling predictive digital twins at scale. *Nature Computational Science*, 1:337–347, 2021.
[3] Mariana Segovia and Joaquin Garcia-Alfaro. Design, modeling and implementation of digital twins. *Sensors*, 22(14), 2022.
[4] John Erkoyuncu, Iñigo Fernández del Amo Blanco, Dedy Ariansyah, Dominik Bulka, Rok Vrabič, and Rajkumar Roy. A design framework for adaptive digital twins. *CIRP Annals*, 69, 05 2020.

[5] Ayman AboElHassan, Ahmed H. Sakr, and Soumaya Yacout. General purpose digital twin framework using digital shadow and distributed system concepts. *Computers & Industrial Engineering*, 183:109534, 2023.

[6] Anastasios Temperekidis, Nikolaos Kekatos, Panagiotis Katsaros, Weicheng He, Saddek Bensalem, Hisham AbdElSabour, Mohamed AbdElSalam, and Ashraf Salem. Towards a digital twin architecture with formal analysis capabilities for learning-enabled autonomous systems. In *MESAS 2022: Modelling and Simulation for Autonomous Systems.*, pages 163–181, 2023.

[7] Sayan Mitra. *Verifying Cyber-Physical Systems: A Path to Safe Autonomy*. The MIT Press, 2021.

[8] Rajeev Alur. *Principles of Cyber-Physical Systems*. MIT Press, 2015.

[9] Cristian Ioan Vasile and Calin Belta. Reactive sampling-based temporal logic path planning. In *2014 IEEE International Conference on Robotics and Automation (ICRA)*, pages 4310–4315, 2014.

[10] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Transactions on Robotics*, 25(6):1370–1381, 2009.

[11] Edward Ashford Lee and Sanjit Arunkumar Seshia. *Introduction to Embedded Systems, Second Edition: A Cyber-Physical Systems Approach*. MIT Press, 2017.

[12] André Platzer. Logic & proofs for cyber-physical systems. In *Proceedings of the 8th International Joint Conference on Automated Reasoning*, pages 15–21, 06 2016.

[13] Marjan Sirjani, Edward A. Lee, and Ehsan Khamespanah. Verification of cyber-physical systems. *Mathematics*, 8(7), 2020.

[14] Georgios Bakirtzis and Ufuk Topcu. Algebraicsystems: Compositional verification for autonomous system design. In *2022 ACM/IEEE 13th International Conference on Cyber-Physical Systems (ICCPS)*, pages 308–309, 2022.

[15] Tichakorn Wongpiromsarn, Mahsa Ghasemi, Murat Cubuktepe, Georgios Bakirtzis, Steven Carr, Mustafa O. Karabag, Cyrus Neary, Parham Gohari, and Ufuk Topcu. Formal methods for autonomous systems. *Foundations and Trends® in Systems and Control*, 10(3-4):180–407, 2023.

[16] Thomas Wright, Cláudio Gomes, and Jim Woodcock. Formally verified self-adaptation of an incubator digital twin. In Tiziana Margaria and Bernhard Steffen, editors, *Leveraging Applications of Formal Methods, Verification and Validation. Practice*, pages 89–109, Cham, 2022. Springer Nature Switzerland.

[17] Edward Lee. Cyber physical systems: Design challenges. *Electrical Engineering and Computer Sciences*, pages 363–369, 06 2008.

[18] Edward A. Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors (Basel)*, 15:4837–4869, 2015.

[19] Murat Cubuktepe, Zhe Xu, and Ufuk Topcu. Distributed policy synthesis of multiagent systems with graph temporal logic specifications. *IEEE Transactions on Control of Network Systems*, 8(4):1799–1810, 2021.

[20] Soroush Bateni, Marten Lohstroh, Hou Seng Wong, Hokeun Kim, Shaokai Lin, Christian Menard, and Edward A. Lee. Risk and mitigation of nondeterminism in distributed cyber-physical systems. In *Proceedings of the 21st ACM-IEEE International Conference on Formal Methods and Models for System Design*, pages 1–11, 2023.

[21] Saad Mubeen, Elena Lisova, and Aneta Vulgarakis Feljan. Timing predictability and security in safety-critical industrial cyber-physical systems: A position paper. *Applied Sciences*, 10(9), 2020.

[22] Aviral Shrivastava, Patricia Derler, Ya-Shian Li Baboudr, Kevin Stanton, Mohammad Khayatian, Hugo A. Andrade, Marc Weiss, John Eidson, and Sundeep Chandhoke. Time in cyber-physical systems. In *2016 International Conference on Hardware/Software Codesign and System Synthesis (CODES+ISSS)*, pages 1–10, 2016.

[23] Robert Davis and Liliana Cucu-Grosjean. A survey of probabilistic timing analysis techniques for real-time systems. *Leibniz Transactions on Embedded Systems (LITES)*, 6:03:1–03:60, 05 2019.

[24] Bineet Ghosh, Clara Hobbs, Shengjie Xu, Don Smith, James H. Anderson, P. S. Thiagarajan, Benjamin Berg, Parasara Sridhar Duggirala, and Samarjit Chakraborty. Statistical verification of autonomous system controllers under timing uncertainties. *Real-Time Systems*, 2024.

[25] Mohammadreza Mehrabian, Mohammad Khayatian, Aviral Shrivastava, Patricia Derler, and Hugo Andrade. A run-time verification method with consideration of uncertainties for cyber–physical systems. *Microprocessors and Microsystems*, 101:104890, 2023.

[26] Stephan Merz. The specification language TLA+. In Dines Bjørner and Martin C. Henson, editors, *Logics of Specification Languages*, pages 401–451. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[27] Heidi Howard, Fritz Alder, Edward Ashton, Amaury Chamayou, Sylvan Clebsch, Manuel Costa, Antoine Delignat-Lavaud, Cedric Fournet, Andrew Jeffery, Matthew Kerner, Fotios Kounelis, Markus A. Kuppe, Julien Maffre, Mark Russinovich, and Christoph M. Wintersteiger. Confidential consortium framework: Secure multiparty applications with confidentiality, integrity, and high availability, 2023.

[28] Igor Konnov, Markus Kuppe, and Stephan Merz. Specification and verification with the TLA+ trifecta: TLC, Apalache, and TLAPS. In *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles*, pages 88–105. Springer International Publishing, 2022.

[29] Paul Regnier, George Lima, and Aline Andrade. A TLA+ formal specification and verification of a new real-time communication protocol. *Electronic Notes in Theoretical Computer Science*, 240:221–238, 2009.

[30] Chris Newcombe, Tim Rath, Fan Zhang, Bogdan Munteanu, Marc Brooker, and Michael Deardeuff. How Amazon Web Services uses formal methods. *Communications of the ACM*, 2015.

[31] Anirban Chaudhuri, Graham Pash, David A Hormuth, Guillermo Lorenzo, Michael Kapteyn, Chengyue Wu, Ernesto ABF Lima, Thomas E Yankeelov, and Karen Willcox. Predictive digital twin for optimizing patient-specific radiotherapy regimens under uncertainty in high-grade gliomas. *Frontiers in Artificial Intelligence*, 6:1222612, 2023.

[32] Matteo Torzoni, Marco Tezzele, Stefano Mariani, Andrea Manzoni, and Karen E. Willcox. A digital twin framework for civil engineering structures. *Computer Methods in Applied Mechanics and Engineering*, 418:116584, January 2024.

[33] L. Lamport. Proving the correctness of multiprocess programs. *IEEE Transactions on Software Engineering*, SE-3(2):125–143, 1977.

[34] Leslie Lamport. The temporal logic of actions. *ACM Trans. Program. Lang. Syst.*, 16(3):872–923, 1994.

[35] Leslie Lamport, John Matthews, Mark Tuttle, and Yuan Yu. Specifying and verifying systems with TLA+. In *Proceedings of the 10th ACM SIGOPS European Workshop*, pages 45–48, 2002.

[36] Alexander T. Ihler, John W. Fisher, III, and Alan S. Willsky. Loopy belief propagation: Convergence and effects of message errors. *Journal of Machine Learning Research*, 6:905–936, 2005.

[37] Claude E. Shannon. A mathematical theory of communication. *Bell System Technical Journal*, 27(3/4):379–423/623–656, 1948.

[38] Yehoshua Bar-Hillel and Rudolf Carnap. Semantic information. *British Journal for the Philosophy of Science*, 4(14):147–157, August 1953.

[39] Stefanie J. Salinger, Michael G. Kapteyn, Cory Kays, Jacob V. R. Pretorius, and Karen E. Willcox. A hardware testbed for dynamic data-driven aerospace digital twins. In Frederica Darema, Erik Blasch, Sai Ravela, and Alex Aved, editors, *Dynamic Data Driven Applications Systems*, pages 37–45, Cham, 2020. Springer International Publishing.

[40] A. Dardenne, S. Fickas, and A. van Lamsweerde. Goal-directed concept acquisition in requirements elicitation. In *Proceedings of the Sixth International Workshop on Software Specification and Design*, pages 14–21, 1991.

[41] Anne Dardenne, Axel van Lamsweerde, and Stephen Fickas. Goal-directed requirements acquisition. *Science of Computer Programming*, 20(1):3–50, 1993.

[42] Heiko Mantel. Information flow and noninterference. In Henk C. A. van Tilborg and Sushil Jajodia, editors, *Encyclopedia of Cryptography and Security*, pages 605–607. Springer, Boston, MA, 2011.

[43] Luke Nelson, James Bornholt, Arvind Krishnamurthy, Emina Torlak, and Xi Wang. Noninterference specifications for secure systems. *SIGOPS Oper. Syst. Rev.*, 54(1):31–39, 2020.

[44] D. McCullough. Noninterference and the composability of security properties. In *Proceedings of the 1988 IEEE Symposium on Security and Privacy*, pages 177–186, 1988.

[45] Boris Köpf and David Basin. An information-theoretic model for adaptive side-channel attacks. In *Proceedings of the 14th ACM Conference on Computer and Communications Security (CCS '07)*, pages 286–296, October 2007.

[46] Lennart Ljung. *System Identification: Theory for the User*. Prentice Hall, 2nd edition, 1999.

[47] Matthew Chan, Nathaniel Snyder, Marcus Lucas, Luis Garcia, Oleg Sokolsky, James Weimer, Insup Lee, Paulo Tabuada, Saman Zonouz, and Mani Srivastava. Let's talk through physics! Covert cyber-physical data exfiltration on air-gapped edge devices. arXiv:2210.07531 [cs.CR]., October 2022.

[48] Siyuan Liu, Ashutosh Trivedi, Xiang Yin, and Majid Zamani. Secure-by-construction synthesis of cyber-physical systems. *Annual Reviews in Control*, 53:30–50, 2022.

[49] S. R. Venkatesh and Munther A. Dahleh. On system identification of complex systems from finite data. *IEEE Trans. Automatic Control*, 46(2):235–257, 2001.

[50] Tuhin Sarkar, Alexander Rakhlin, and Munther A. Dahleh. Finite time LTI system identification. *Journal of Machine Learning Research*, 22:1–61, 2021.

[51] Dalton Jones and Munther A. Dahleh. Closed loop system identification with known feedback: A non asymptotic viewpoint. In *Proceedings of the 2022 American Control Conference (ACC)*, June 2022.

[52] Edmund M. Clarke, William Klieber, Miloš Nováček, and Paolo Zuliani. Model checking and the state explosion problem. In Bertrand Meyer and Martin Nordio, editors, *Tools for Practical Software Verification*, pages 1–30. Springer, Berlin, Heidelberg, 2012.

[53] Alon Brook, Doron Peled, and Sven Schewe. Local and global fairness in concurrent systems. In *2015 ACM/IEEE International Conference on Formal Methods and Models for Codesign*, pages 2–9, 2015.

[54] Rob Van Glabbeek and Peter Höfner. Progress, justness, and fairness. *ACM Comput. Surv.*, 52(4), 2019.

[55] Matt Luckcuck, Marie Farrell, Louise A. Dennis, Clare Dixon, and Michael Fisher. Formal specification and verification of autonomous robotic systems: A survey. *ACM Comput. Surv.*, 52(5), 2019.

[56] Matthias Rungger and Paulo Tabuada. Abstracting and refining robustness for cyber-physical systems. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*, pages 223–232, 2014.

[57] Vinith Misra, Vivek K Goyal, and Lav R. Varshney. Distributed scalar quantization for computing: High-resolution analysis and extensions. *IEEE Transactions on Information Theory*, 57(8):5298–5325, August 2011.

[58] Phillip Stanley-Marbell, Armin Alaghi, Michael Carbin, Eva Darulova, Lara Dolecek, Andreas Gerstlauer, Ghayoor Gillani, Djordje Jevdjic, Thierry Moreau, Mattia Cacciotti, Alexandros Daglis, Natalie Enright Jerger, Babak Falsafi, Sasa Misailovic, Adrian Sampson, and Damien Zufferey. Exploiting errors for efficiency: A survey from circuits to applications. *ACM Comput. Surv.*, 53(3):51, May 2021.

[59] Nancy A. Lynch. *Distributed Algorithms*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1996.

[60] Leslie Lamport. On interprocess communication. *Distributed Computing*, 1985.

[61] Daniel Jackson and Eunsuk Kang. Property-part diagrams: A dependence notation for software systems. *IEEE 31st International Conference on Software Engineering*, 01 2009.

A BACKGROUND ON THE PGM FRAMEWORK AND TLA

A PGM encodes random variables as nodes and statistical dependencies as edges between nodes. In the DT PGM framework, the PGM governs how variables are updated in each timestep. An edge between two nodes dictates the dependence of the destination node on the parent node. For example, the edge $S \rightarrow O$ represents that observational data O depends on the physical state S.
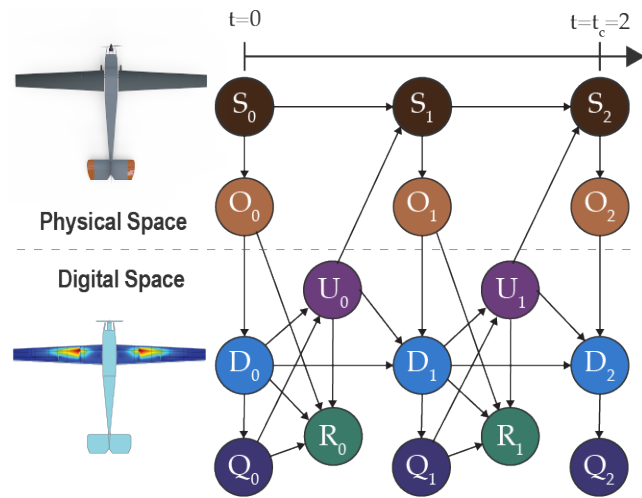

Figure 12: Probabilistic graphical model (PGM) describing the UAV DT.

B STATE MACHINE FORMALIZATION

This section details additional assumptions of the finite state machine formalization.

Fairness. Fairness is crucial in scheduling processes within the state machine framework. It ensures all processes are treated justly under the operational rules of the system. Specifically:

$$\mathcal{F} = \{UF(\omega_i) \lor WF(\omega_i) \lor SF(\omega_i) \mid \omega_i \in \Omega\} \quad (14)$$

Here, (14) categorizes each process in the process set Ω into three types: Unfair (UF), where it is allowable that a process may never execute; Weakly Fair (WF), which guarantees that if a process is continuously enabled, it will eventually execute; and Strongly Fair (SF), requiring that if a process is enabled intermittently, it must eventually be executed. The classification into these categories depends on the system's dynamic requirements and stakeholder inputs, guiding how processes are triggered during the state machine's operation. For instance, a process like $S \rightarrow S$, representing the continuous update of the UAV's physical state, is deemed Strongly Fair because its execution is essential and inevitable, reflecting the continuous nature of physical state updates. Weak fairness is generally assumed and warranted for many real-world systems [53, 54] as it is unrealistic that a process can wait indefinitely before executing.

Initial and Termination States. The initial state of a state machine is crucial as it sets the baseline from which all processes begin. For the DT, this is defined by a logical predicate that assigns a starting value to each variable in the system. This predicate ensures all components of the DT start in a well-defined state that is consistent with the expected initial conditions of its physical twin. For instance, if modeling a UAV DT, the initial state may specify the UAV's starting health, the initial digital state, etc.

While a DT can theoretically operate indefinitely, practical applications often require defining specific conditions under which the simulation or operation should cease. This is termed the termination state $\mathcal{T}$, which is also expressed as a logical predicate. Termination conditions can vary widely depending on the system's purpose but generally include achieving a goal, exhausting resources, or encountering a specific event that requires halting operations. For example, a UAV's digital twin might terminate when the UAV completes its mission objectives or when the digital twin finishes program execution for a specified duration.

B.1 Abstraction of Digital Twin complexity

In developing the state machine for the DT, our aim was not only to capture the dynamic interplay of components but also to abstract complex DT behaviors into manageable and verifiable forms. This abstraction focuses on simplifying intricate component behaviors—whether physical phenomena or computational complexities—while preserving the essential characteristics necessary for accurate system modeling. Here, we outline our methodology for abstracting these complexities, providing general principles that are later applied in specific contexts within Sec. 4.

Asserting existence instead of computing numerical operations One fundamental principle in formal methods is to describe the outcomes of system operations rather than detailing the specific computations that achieve these outcomes [35, 55, 56]. In line with this principle and consistent with prior literature in other domains [57, 58], our approach simplifies the numerical complexity inherent in DT operations, yet maintains the integrity of the computational results in the abstracted state machine actions.

Retain probabilistic characteristics In our methodology, simplification does not eliminate stochastic behavior inherent to many DT processes, especially those involving predictive, probabilistic components. However, instead of quantifying specific probabilities, our abstraction focuses on delineating possible

behaviors by writing indeterministic actions in TLA [35].

Representing a distributed message channel as a queue with deterministic write and nondeterministic read Recall from Sec. 3 that we represent a messaging channel as a random variable in the augmented PGM. Following the established practice in formal methods of using queues to represent network communication [59], we abstract each message channel random variable as a queue, where messages are 'pushed' onto the queue as they are sent and 'popped' from the queue in a nondeterministic order. This reflects potential real-world communication issues like delays, losses, or reordering, simplifying the analysis by eliminating the need to track details such as the timing specifics of each message.

$$
\begin{aligned}
&\text{channel} \leftarrow \text{Queue}[] \\
&\quad \text{push } x := \text{append channel}, x \\
&\quad\ \text{pop } i := \text{channel}[i] \\
&\text{remove } i := \{\text{channel}[i] \in 1..\text{Length(channel)} \mid i \neq j\} \\
&\qquad\qquad \text{where } i \in [1, \eta] \\
&\qquad\qquad\ \text{with } SF(i = 1)
\end{aligned}
\tag{15}
$$

Index i ranges from 1 to η, a parameter representing out-of-order message delivery constraints. For instance, a realistic and common constraint in wireless networks is a limited buffering and processing window. Index i is randomly chosen, where $i = 1$ indicates that the correct, most recent message is being delivered. We impose the strong fairness condition $SF(i = 1)$ to ensure the correct message will eventually be delivered, a realistic expectation that mirrors guarantees provided by many network protocols such as Bluetooth.

Atomicity in system orchestration Atomicity is a fundamental consideration in formal methods, particularly when defining specifications for complex systems [60]. In our context, two processes are deemed mutually atomic when their state transitions are considered simultaneous and uninterruptible, essentially occurring within the same atomic slice of time. In designing the orchestration for the DT, we strategically treat certain groups of operations as atomic to simplify the model and enhance tractability. This decision helps manage complexity by reducing the granularity of interaction between components, focusing on high-level system behavior rather than the minutiae of inter-process communication.

B.2 Specifying Properties

In the process of specifying properties, initial analysis identifies two critical aspects of synchronization: (1) Consistency between the digital and physical states, and (2) Execution of control commands by the physical twin as issued by the DT. These insights lead to the development of properties P_2 and P_7, visualized in Fig. 8. Further decomposition of P_2 reveals that the digital state's update relies on consistent sensor data and computed control inputs, reflected in property P_4: *Sensor and control inputs to update digital state must be consistent.* We observe that for consistent sensor inputs to be feasible, the DT must effectively receive and process incoming sensor data, a requirement captured in property P_5. This analysis continues, breaking down each goal into finer-grained sub-properties, eventually organizing them using goal structuring notation [61].

Correct orchestration naturally requires component correctness as well. For example, property P_3 articulates that the predictive model within the digital twin must be correct to ensure that the digital state accurately twins the physical state over time. We describe these critical functionalities as explicit sub-properties within our framework. For instance, we configure our design to simulate scenarios where P_3 is false, allowing us to explore and verify the system's behavior under conditions of component failure. This transformation of potentially implicit assumptions into concrete, testable elements within our specification not only crystallizes the verifiable state of each component but also elucidates their role and relationship to the system's overall behavior.

C TLA CODE

The TLA code is written and model-checked with the TLA+ toolkit. Fig. ?? shows a portion of this code, specifically modeling the procedure `DT_ReceiveObsDelayed(s, m)`. This process represents actions taken for a specific sensor `s` with a particular delay index `m`.

In this snippet, `DT_ReceiveObsDelayed` checks if there are any queued observations for sensor `s` (by verifying `n_obs[s]` is not empty). If observations exist, it then checks if the delay index `m` corresponds to an entry within the domain of `n_obs[s]`. If so, it evaluates whether the timestamp `n_obs[s][m]["t"]` is recent enough, as determined by the function `OM!IsMessageUpToDate`. If the message is up-to-date, it appends this observation to the list of received observations `obs_in[s]` and removes it from the observation queue `n_obs[s]`. If the timestamp is outdated, the entry is removed from `n_obs[s]`, leaving `obs_in[s]` unchanged. In cases where the delay index `m` is not present, both `obs_in[s]` and `n_obs[s]` remain unchanged.

The full TLA+ specification, including this process and others within the DT model, is published on GitHub.

1393
1394
1395
1396
1397
1398
1399
1400
1401
1402
1403
1404
1405
1406
1407
1408
1409
1410
1411
1412
1413
1414
1415
1416
1417
1418
1419
1420
1421
1422
1423
1424
1425
1426
1427
1428
1429
1430
1431
1432
1433
1434
1435
1436
1437
1438
1439
1440
1441
1442
1443
1444
1445
1446
1447
1448
1449
1450
1451
1452
1453
1454
1455
1456
1457
1458
1459
1460
1461
1462
1463
1464
1465
1466
1467
1468
1469
1470
1471
1472
1473
1474
1475
1476
1477
1478
1479
1480
1481
1482
1483
1484
1485
1486
1487
1488
1489
1490
1491
1492
1493
1494
1495
1496
1497
1498
1499
1500
1501
1502
1503
1504
1505
1506
1507
1508