Machine Dynamics-aware CPS Scheduling and Iterative Validation for Industry 4.0

Pietro Turco*, Enrico Fraccaroli*†, Samarjit Chakraborty†, Franco Fummi*
*Department of Engineering for Innovation Medicine, University of Verona, Italy, name.surname@univr.it
†Dept. of Computer Science, University of North Carolina at Chapel Hill, USA, enrifrac|samarjit}@cs.unc.edu

***Abstract*—Modern manufacturing demands physically feasible, optimized schedules, but traditional methods often neglect machine dynamics, yielding theoretically optimal yet not robustly feasible plans in practice. This paper presents a comprehensive workflow bridging this gap for cyber-physical production systems. Our first contribution enhances production line schedule synthesis with variable discretization steps for mode sequences, balancing accuracy and efficiency. Second, we propose a novel validation framework using Frost, a deterministic digital twin built on Lingua Franca. This framework models physical dynamics using adaptive ODE solvers and the full software/communication stack. Crucially, significant deviations in Frost simulations trigger an iterative re-synthesis loop. This closed-loop approach, underpinned by Lingua Franca's determinism, ensures schedules are rigorously validated and iteratively improved for practical feasibility in complex cyber-physical settings.**

***Index Terms*—Cyber-Physical Systems, Digital Twin, Lingua Franca, Scheduling, Simulation-based Validation, Hybrid Systems, Manufacturing Systems, Model-Based Design**

I. Introduction

The advent of Industry 4.0 has introduced unprecedented adaptability into Cyber-Physical Production Systems (CPPSs). Modern manufacturing systems feature reconfigurable cells, smart sensors, and high integration between physical processes and control software, necessitating flexible scheduling solutions adaptable to varying workloads and production goals. In this paper, *scheduling* defines the process of determining operation sequences and precise timing to achieve specific production objectives.

Traditional scheduling methods often treat jobs as atomic operations with fixed durations [1], overlooking the rich physical dynamics of manufacturing equipment. Recent work, such as [2], [3], has addressed this by incorporating underlying physical processes into scheduling formulations, treating manufacturing steps as hybrid systems governed by mode-specific differential equations. Their two-stage optimization framework uses BFS for sequence identification and PSO for switching instant refinement, balancing combinatorial complexity with continuous parameter precision.

However, these dynamics-aware approaches typically rely on fixed discretization steps for all modes and stop short of validating schedules in a comprehensive system-level simulation. Generated schedules are often tested in isolation, neglecting real-time semantics, component interactions, or platform-level behavior. This creates a disconnect between theoretical optimization and practical execution.

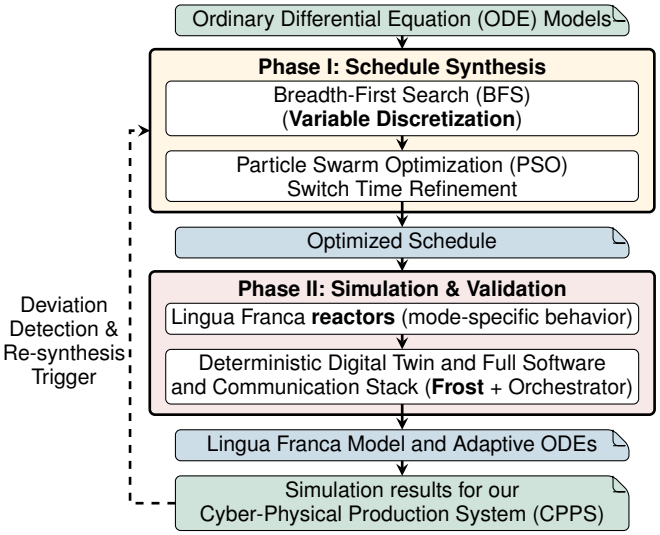

Figure 1: Simulation-driven workflow: from ODE-based hybrid models to optimized schedules, executed as Lingua Franca reactions in the Frost platform for system-level validation.

Contributions of this paper: This paper presents a comprehensive, simulation-driven workflow designed to bridge the gap between idealized schedule synthesis and practical execution in cyber-physical production systems. Our work introduces several key contributions:

First, we significantly enhance the schedule synthesis process. Unlike prior works that relied on fixed discretization steps, we incorporate **variable discretization steps** for mode sequences. This allows for more accurate and computationally efficient exploration of the solution space, adapting temporal resolution to specific mode dynamics.

Second, we propose a **novel validation framework** built upon Frost, a deterministic digital twin platform leveraging Lingua Franca. While traditional validation often focuses solely on physical models, our framework meticulously models the **full software and communication stack** of the system, integrating adaptive step ODE solvers with zero-crossing detection for high-fidelity physical dynamics. This provides a level of system-level fidelity and determinism not typically found in existing co-simulation environments.

Figure 1 shows our workflow. It begins with enhanced synthesis generating dynamics-aware schedules. These are then rigorously validated within Frost, where feasibility and robustness are assessed against a realistic digital twin environ-

ment. When Frost simulation reveals significant deviations, it triggers a **iterative refinement loop** and a re-synthesis. This closed-loop approach, underpinned by Lingua Franca's unique determinism and superdense time semantics, ensures schedules are rigorously validated and iteratively improved for feasibility.

II. Related Work

Dynamics-aware scheduling integrates physical system behavior into optimization. Works like [2], [3] pioneered synthesizing schedules for hybrid systems with mode-dependent dynamics, using BFS for sequence exploration and PSO for refinement. Our work extends this by introducing **variable discretization steps** in synthesis, enhancing accuracy and efficiency by adapting temporal resolution to mode dynamics.

Beyond synthesis, schedule validation in realistic cyber-physical environments is crucial. Traditional co-simulation tools and standards, such as FMI [4], Modelica, or Simulink, excel at physical model interoperability. However, their master algorithms and handling of concurrent events can lead to non-deterministic behavior. Digital twin concepts [5] mirror physical assets but often lack formal guarantees for complex software interactions and communication protocols. Our approach leverages Frost, a platform built on Lingua Franca [6]. This provides a **deterministic digital twin** that integrates high-fidelity physical dynamics (via **adaptive step ODE solvers with zero-crossing detection**) and, uniquely, meticulously models the **full software and communication stack** of the manufacturing system. This focus on deterministic coordination of software interactions addresses a critical gap in traditional physical-centric co-simulation.

The concept of iterative design and closed-loop optimization is established, but its application to end-to-end dynamics-aware schedule synthesis and validation in Cyber-Physical Systems (CPSs) with formal feedback remains an active area. Our proposed **iterative refinement loop**, triggering re-synthesis upon detecting significant deviations in the digital twin, offers a novel closed-loop approach for robust feasible schedules, moving beyond one-shot optimization.

III. Motivating Example and Problem Setup

Many manufacturing operations, such as drilling or tapping, involve physical processes whose behavior depends on machine configuration and material interaction. To illustrate the scheduling challenges under such dynamics, we begin with a concrete example involving a multi-mode machining task.

A. Motivating Example

Consider a CNC drilling operation where a machine drives a spindle to a specified depth across a layered workpiece (see Figure 2). The machine supports multiple operating modes, each with a distinct speed-torque profile (e.g., high-speed for soft material, high-torque for denser sections to avoid tool wear). Switching modes incurs costs (energy, transient behavior), and operation duration depends nonlinearly on the selected mode and material. This example illustrates manufacturing steps where machine mode affects physical execution and scheduling. Our framework tackles the complexity of deterministic modeling and validation of interactions, control, and communication within a CPS, challenging traditional fixed-duration schedulers.

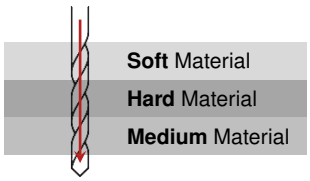

Figure 2: A Computerized Numerical Control (CNC) drill.

B. Problem Setup

We generalize this scenario to manufacturing tasks requiring physical processes under mode-dependent dynamics. Let $\mathcal{M} = \{m_1, m_2, \ldots, m_N\}$ be the set of available machine modes. Each mode m_i induces continuous-time behavior modeled by:

$$\dot{x}(t) = f_i(x(t), u(t)), \quad t \in [t_k, t_{k+1}) \,. \tag{1}$$

Here, $x(t)$ is the system state, $u(t)$ is a control input, f_i defines the Ordinary Differential Equations (ODEs) governing dynamics in mode m_i, and $[t_k, t_{k+1})$ is the time interval for mode m_i. Index i refers to the mode, and k to the discrete time step. This modeling reflects physical evolution commonly captured by ODEs.

Objective: Given an initial state and task goal, the objective is to synthesize a schedule that selects a sequence of modes and switching times, minimizes execution time and energy consumption, and ensures feasibility under physical and operational constraints. While synthesis optimizes for efficiency, validation within the digital twin verifies safety and prevents undesirable states.

Constraints: The synthesis must consider transition costs and delays between modes, dynamic feasibility (e.g., torque or thermal limits), and discretization granularity. These constraints can be formalized as predicates on $x(t)$ or mode transitions. This formulation captures the hybrid nature of operations, where high-level scheduling interacts tightly with continuous physical behavior. For drilling, constraints include avoiding overheating and timely completion of operations.

IV. Dynamics-Aware Schedule Synthesis

This section builds upon the dynamics-aware scheduling formulation from [2], synthesizing mode schedules by reasoning over the system's physical dynamics, and extends it by introducing variable discretization steps for each mode.

A. Switched System Model

Each manufacturing task is modeled as a switched linear system, where each mode corresponds to a different set of continuous-time dynamics. We focus on a class of switched linear systems for tractable search and optimization.

Each mode m_i is characterized by linear dynamics:

$$\dot{x}(t) = A_i x(t) + B_i u(t), \quad \text{for } t \in [t_k, t_{k+1}) \,. \tag{2}$$

This is a specific case of Equation (1), capturing key mode-dependent behaviors for efficient synthesis. Here, (A_i, B_i) are system matrices for mode m_i, and $u(t)$ is a constant or mode-specific input over $[t_k, t_{k+1})$. The goal is to synthesize a feasible sequence of mode activations and switching instants

to drive the system from x_0 to a goal region x_{goal}, representing target physical conditions (e.g., depth, torque).

B. Discretization and Search Space Construction

To enable combinatorial search, we discretize the continuous-time dynamics using an exact step-wise formulation. While prior work often relied on a single, fixed time step h for all modes, our work introduces **variable discretization steps** for each mode. This allows the synthesis engine to adapt the temporal resolution to the specific dynamics of the active mode. For a given mode m_i, the discretization step can be denoted as h_i. This means more dynamic modes can utilize a finer h_i, while more static modes can use a coarser h_i to reduce the search space without significant loss of fidelity. This variable h_i can be determined based on the mode's inherent dynamic characteristics or as a configurable parameter, and can be **iteratively refined** based on feedback from the validation phase (see Section V-D).

Assuming a piecewise-constant control input u_k over each **mode-specific** intervals of duration $h_i = t_{k+1} - t_k$, the state update under mode m_i is given by:

$$x_{k+1} = e^{A_i h_i} x_k + \left(\int_0^{h_i} e^{A_i \tau} B_i \, d\tau \right) u_k \, . \tag{3}$$

This expression follows from the variation of constants formula for linear time-invariant systems. Letting $\Phi_i = e^{A_i h_i}$ and $\Gamma_i = \int_0^{h_i} e^{A_i \tau} B_i \, d\tau$, we rewrite the update as:

$$x_{k+1} = \Phi_i x_k + \Gamma_i u_k \tag{4}$$

Each discrete step is assumed to maintain the current mode for the duration of its specific time step h_i. Allowing us to represent a schedule as a finite sequence of modes applied over successive discrete time intervals, each with its own duration.

We explore the space of discrete mode sequences using a Breadth-First Search (BFS) tree to construct candidate schedules. Each node corresponds to a reachable state x_k produced by executing a mode sequence. Children are generated by simulating all valid mode transitions for one time step. Without explicit constraints, we assume all modes are reachable from any other. The engine can encode transition restrictions as a feasibility predicate during children node expansion. In the drilling example, we can restrict switches to be only between sequential speed-torque profiles to avoid abrupt changes.

C. Cost Modeling and Pruning

To evaluate candidate schedules, we define a multi-objective cost function capturing execution time and energy consumption. We use k to index discrete scheduling steps, and m_k for the mode selected at step k. During the discrete BFS phase, energy is approximated using surrogate cost terms:

$$J = \sum_{k=1}^{K} \left(\alpha \cdot h_i + \beta \cdot x_k^\top Q_{m_k} x_k \right) \tag{5}$$

where α and β are weighting factors for tuning the trade-off between execution time and energy consumption, and Q_{m_k} is a positive semidefinite matrix encoding an energy profile for mode m_k (see Equation (11) in [3]). The term $\alpha \cdot h_i$ captures time cost, while the quadratic form encodes mode-specific energy behavior. This formulation is for the discrete-time setting, where energy is approximated. The methodology also supports numerical integration of continuous dynamics during the search for more accurate trajectory and cost evaluation.

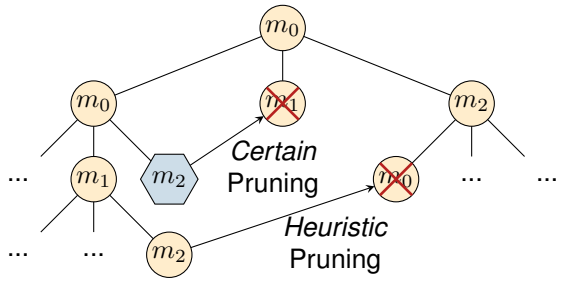

Figure 3: Illustrative BFS search tree and pruning. Nodes represent reachable states under different mode sequences.

To manage combinatorial explosion, we prune the BFS tree using two strategies. The **first**, *state similarity heuristics*, discards nodes that reach approximately equivalent states with higher cost. These heuristics are model-dependent, tailored to exploit system characteristics for efficient pruning. For instance, in the drilling scenario, a heuristic might consider two solutions approximately equivalent if they reach similar depths with comparable resource consumption, prioritizing the one with lower cost. The **second**, *pareto front filtering*, retains only non-dominated solutions in the cost space at each depth. This efficiently retains high-quality candidates while discarding suboptimal branches early. Variable discretization steps, by providing more accurate mode dynamics representation, further enhance pruning effectiveness.

Figure 3 exemplifies a possible BFS search. Orange nodes represent incomplete solutions, while blue hexagonal ones are complete solutions, red crossed nodes are pruned ones. When comparing a complete solution against an incomplete one, the cost of the complete solution provides a definitive lower bound. When comparing multiple incomplete solutions, heuristics estimate which branch is more promising.

D. Switching Time Refinement via PSO

While the BFS-based synthesis efficiently generates feasible mode sequences, the resulting switching times may still be suboptimal due to heuristic pruning or temporal discretization. To improve fidelity, we refine switching times using Particle Swarm Optimization (PSO), following [2]. This two-stage approach, where BFS handles combinatorial sequence search and PSO refines continuous parameters, balances tractability and precision. Each particle encodes a vector of mode durations $\{\Delta t_1, \Delta t_2, \ldots, \Delta t_K\}$ corresponding to candidate schedule segments. These durations are continuous variables bounded within a range centered around values from the BFS phase. The PSO objective function mirrors the previous model but leverages continuous integration for precise energy and timing evaluation. Particles are iteratively updated based on local and global best solutions, converging when improvements fall below a threshold or a max-

imum number of iterations is reached. Consider a CNC drilling operation where a machine drives a spindle to a specified depth across a layered workpiece (see Figure 2). Figure 4 illustrates this behavior. This refinement enables higher-resolution switching profile tuning while preserving the selected mode sequence. Prior work [2] observed energy consumption consistently reduced (typically 4–5%, max 6.5%) and execution time improved (up to 3%) in favorable cases. Even with slight execution time worsening (up to 3%), energy savings often justify the trade-off.

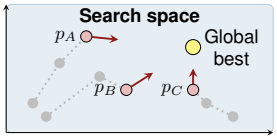

Figure 4: PSO search behavior.

V. From Schedule to Simulation in Frost

Schedules from enhanced synthesis (variable discretization, PSO refinement) serve as input for validation. This synthesis is iterative: if the digital twin reveals significant deviations, the synthesis engine can be re-invoked with updated information for iterative refinement. We deploy these schedules within Frost, a modular Lingua Franca-based platform, for validation in a realistic cyber-physical environment. Frost enables time-coordinated simulation, preserving continuous-time dynamics while ensuring deterministic logical execution. This section provides Lingua Franca and Frost background, then presents our synchronization algorithm.

A. Lingua Franca: Deterministic Coordination with Superdense Time

Lingua Franca is a polyglot coordination language designed to guarantee deterministic execution of reactive concurrent components, known as reactors [6]. Reactors are stateful components that communicate via input/output ports and can declare time-triggered and event-triggered behaviors [7]. Code is encapsulated within reactions, executed in response to events or input activity [8]. A key feature is **superdense time**, a pair of timestamp and integer microstep [7]. The timestamp indicates logical execution time, while the microstep distinguishes multiple events at the same logical time [6], guaranteeing deterministic execution order.

Another important feature is built-in detection of **causality loops** [9], arising from cyclic triggers between reactors, potentially stalling simulation. Lingua Franca automatically analyzes the dependency graph and detects cycles at compile time. It suggests introducing a logical delay on output ports to advance superdense time and break the cycle. This mechanism is essential for modeling realistic CPSs with bidirectional communication and feedback, ensuring correct and deadlock-free simulations. Superdense time and Lingua Franca determinism are fundamental for our orchestration approach, ensuring physically simulated models interact deterministically and reliably, even with simultaneous activities. Determinism is formally guaranteed by its underlying semantics (e.g., [6], [7]).

B. Frost: A Modular CPS Simulation Platform

Frost, previously introduced in [10], is an open-source simulation framework built on Lingua Franca for industrial CPS simulation. It provides a flexible, modular, and extensible environment for modeling, simulating, and analyzing complex industrial systems. Frost achieves this through abstractions that form a message-driven, service-oriented infrastructure. Its modular architecture allows users to implement custom components (sensors, actuators, controllers, machines) and integrate them seamlessly. At its core, Frost separates machine description into a data model (exposed machine data and services, organized in a tree-like structure with nodes for state and methods) and component behavior (implemented in Lingua Franca, defining how components react to events).

Frost communication infrastructure is composed of a `FrostBus` (message broker) and a `FrostMachine` (service provider), enabling efficient and coordinated data exchange. `FrostMachine` processes requests and triggers logic via the data model. Frost provides a comprehensive simulation platform that hides the communication infrastructure, allowing users to focus on system modeling and simulation.

C. Coordinated Simulation via Logical-Time Orchestration

CPSs involve diverse components with varying time scales. To manage this, we developed a flexible simulation algorithm that adapts to each component's temporal behavior, ensuring synchronous execution and deterministic interaction while maintaining high-fidelity physical simulation. First, each system's ODE solver performs a dry run, proposing a variable and event-aware time step h (e.g., stopping at zero-crossings). Then, the orchestrator collects these proposed time steps and computes the minimum: $h_m = \min(h)$. Finally, each component then performs a second simulation pass using h_m and updates its internal state.

To implement this algorithm in Frost, we extended the `FrostMachine` component. Listing 1 shows the orchestrator's Lingua Franca code. It holds simulation state, manages time advancement, and coordinates simulation phases. Upon receiving messages (dry run or simulation responses), it computes

Listing 1: Lingua Franca code of the Orchestrator.

```
reactor Orchestrator(...) extends FrostMachine{
  state models = [...]
  state pending_responses = len(models)
  state responses = []
  state time_step = 1.0
  logical action advance
  reaction(channel_in) -> channel_out, advance{=
    for message in channel_in[0].value:
      if message.payload == "dry_run":
        responses.append(message.result)
      pending_responses -= 1
    if pending_responses == 0:
      if responses:
        time_step = min(responses)
        channel_out[0].set(build_broadcast_message("
            simulate"))
      else:
        advance.schedule(time_step)
      pending_responses = len(models)
      responses = []
  =}
  reaction(advance) -> channel_out{=
    channel_out[0].set(build_broadcast_message("dry_run")
        )
  =}
}
```

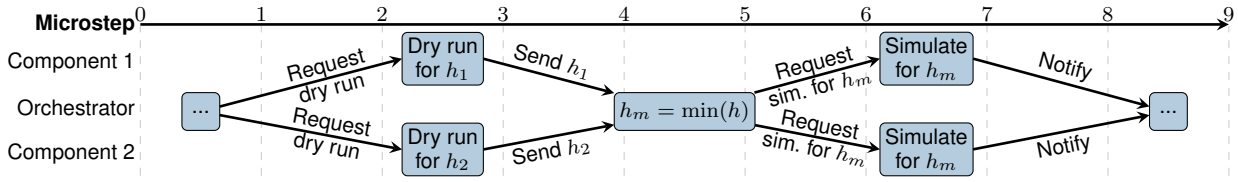

Figure 5: Execution flow of the simulation algorithm (there is a 1 microstep delay due to the bus).

the minimum time step and schedules broadcast requests or advances logical time. This mechanism, underpinned by Lingua Franca's superdense time semantics, enforces deterministic, time-aligned progression across all heterogeneous models. The ability to integrate adaptive step solvers with event detection significantly enhances physical simulation fidelity while maintaining strong determinism guarantees.

Figure 5 illustrates the simulation flow. The orchestrator and components operate within the same logical timestamp, with actions offset by successive microsteps. This algorithm leverages superdense time to compute the next step within the same logical time instant. The communication delay (2 microsteps) is due to causal loops (see Section V-A) and messages passing through the `FrostBus`. This ensures all components are synchronized and the platform follows their simulation speed.

D. Deviation Detection and Iterative Refinement

The validation process within Frost identifies discrepancies between the synthesized plan and simulated reality, crucial for iterative refinement. During simulation, the framework monitors the system's state and behavior for significant deviations or constraint violations.

Deviation detection occurs via: i) **State Trajectory Mismatch**, by comparing synthesized (S_i) vs. simulated (S'_i) state variables ($x(t)$), where a significant deviation ($|S_i - S'_i|$ exceeding a tolerance) signals a need for re-synthesis, often indicating that synthesis discretization steps (h_i) were too coarse; ii) **Constraint Violations**, by detecting violations of physical or operational constraints not fully captured in synthesis (e.g., torque limits, unsafe states, tool wear); and iii) **Unmodeled Events**, by identifying unexpected events from detailed software/communication stack modeling (e.g., communication timeouts, API failures, concurrent interactions) absent in the abstract synthesis model.

Upon detecting a significant deviation, particularly a state trajectory mismatch, the validation triggers a re-synthesis in Phase 1. Feedback to the synthesis engine includes the deviation's nature, time, state, and suggestions to refine discretization steps for problematic modes/segments. This closed-loop mechanism iteratively generates more robust and practically feasible schedules, addressing real-world complexities.

VI. Framework Demonstration: the ICE Lab

This section demonstrates our framework's application, combining enhanced schedule synthesis and a deterministic digital twin, to a manufacturing scenario in the ICE Laboratory [11]. This case study involves a simulation *Orchestrator* (see Section V-C), a *Milling Machine*, and a *Robotic Arm* connected via a FrostBus. Despite a simplified setup, Frost's modularity and deterministic coordination ensure seamless scaling to complex industrial scenarios.

The workflow starts with enhanced schedule synthesis (see Section IV) to generate dynamics-aware schedules, utilizing **variable discretization steps** for tailored mode sequences and switching times (e.g., finer steps for high-dynamic drilling, coarser for steady-state). This schedule then feeds into the Frost digital twin for validation. Machine reactors (e.g., Milling machine, Listing 2) extend `FrostMachine`, and the simulation executes these models, integrating **adaptive step ODE solvers with zero-crossing detection** for high-fidelity physical dynamics and precise event detection.

Crucially, validation also covers the **full software and communication stack**. Machine interactions (e.g., Milling machine and robotic arm), including API calls and FrostBus messaging, are simulated with inherent delays and concurrency. This reveals issues physical-only simulations miss, such as timing violations from communication latency, unexpected behavior from concurrent API calls or race conditions, and constraint violations (e.g., tool overheating) with realistic software delays.

Listing 2: Pseudocode representing the Milling machine ODE model inside Frost.

```
target python
preamble {= from drill_ode import run_sim_step =}
reactor MillingMachine(
  V=9.6,R=8.4,L=0.0084,J=0.01,Kd=0.25,Ke=0.1785,
  Kt={=141.6*0.1785=},Fd=0.064,Fs=0.035,Ts=1.0,
  Gr=20.0,ThR=2.2,ThC={=9.0/2.2=},ThAmb=22.0,
  target_depth=10.0) extends FrostMachine{
  state x # [I, w, depth, T]
  method dry_run_method(ts){=
    params = (self.V, self.R, self.L, self.J,
      self.Ke, self.Kt, self.Kd, self.Fd, self.Gr,
      self.Fs, self.Ts, self.ThC, self.ThR, self.ThAmb)
    return run_sim_step(
      self.x, ts, params, self.target_depth, True)
  =}
  method simulate_method(ts){=
    params = (self.V, self.R, self.L, self.J,
      self.Ke, self.Kt, self.Kd, self.Fd, self.Gr,
      self.Fs, self.Ts, self.ThC, self.ThR, self.ThAmb)
    self.x = run_sim_step(
      self.x, ts, params, self.target_depth, False)
    return True
  =}
  reaction(startup) {=
    data_model.get_node(".../dry_run").callback = self.
        dry_run_method
    data_model.get_node(".../simulate").callback = self.
        simulate_method
  =}
}
```

As shown in Listing 2, the Lingua Franca reactor for the Milling Machine leverages an external Python module to implement its continuous-time dynamics and simulation logic.

This separation allows for a concise Lingua Franca model while encapsulating the complex numerical integration details.

Listing 3: Python code of the Milling Machine's ODE model, event detection, and combined simulation step logic.

```
import numpy as np
from scipy.integrate import solve_ivp
def dxdt(t,x,V,R,L,J,Ke,Kt,Kd,Fd,Gr,Fs,Ts,ThC,ThR,ThAmb):
  I, w, D, T = x
  dx0 = -(R/L)*I-(Ke/L)*w+(V/L)
  dx1 = +(Kt/J)*I-(Kd/J)*w-((Fd*Gr)/J)*D-(Gr/J)*Fs
  dx2 = ((Ts*Gr)/(2*np.pi))*w
  dx3 = (R/ThC)*I**2+(ThAmb-T)/(ThC*ThR)
  return [dx0, dx1, dx2, dx3]
def depth_event(t, x, V, R, L, J, Ke, Kt, Kd, Fd, Gr, Fs,
                Ts, ThC, ThR, ThAmb, target_depth):
  return x[2] - target_depth
depth_event.terminal = True
depth_event.direction = 1
def run_sim_step(x, ts, params, target_depth, dry_run):
  initial_state = np.array(x)
  t_span = [0, ts]
  event_params = params + (target_depth,)
  depth_event.args = event_params
  sol = solve_ivp(dxdt, t_span, initial_state,
    args=params, events=(depth_event if dry_run else None
        ),
    dense_output=False)
  if not dry_run:
    return sol.y[:, -1].tolist()
  if sol.t_events[0].size > 0:
    return sol.t_events[0][0]
  return ts
```

As shown in Listing 3, the `derivatives` function defines the system's ODEs, while the `depth_event` function handles zero-crossing detection. The `run_simulation_step` function orchestrates the numerical integration using SciPy's `solve_ivp`, performing either a dry run (for event detection) or a full simulation step (for state advancement). This allows for precise simulation of physical behaviors, such as tool wear or material deformation, and accurate detection of events like reaching a specific depth or exceeding a force threshold.

As detailed in Section V-D, the framework incorporates a **deviation detection and iterative refinement** mechanism. If, during the Frost simulation, the actual depth of the drill deviates significantly from the synthesized trajectory due to unmodeled friction, or if a communication delay causes a critical timing violation, the framework *would detect* this deviation. This detection *would then trigger* a re-synthesis (looping back to Section IV), potentially with updated constraints, refined models of the problematic phase, or a modified objective function, to generate a more robust and practically feasible schedule. This iterative process ensures that the final schedule is resilient to the complexities of the real-world cyber-physical environment, leading also to significant performance improvements. Our framework determined schedules that, on average, achieved 35% better performance than naive single-mode algorithms, with some solutions demonstrating improvements of up to 53.8% in overall Pareto front quality. Furthermore, the iterative refinement (PSO) step yielded additional benefits, leading to manufacturing times reduced by up to 57% and energy consumption by up to 75% compared to these baseline methods.

VII. Conclusion

This paper presented a comprehensive, simulation-driven workflow bridging idealized schedule synthesis and real-world CPSs. It introduced an enhanced synthesis method with **variable discretization steps** and a novel validation framework built upon Frost, a **deterministic digital twin** modeling high-fidelity physical dynamics and the **full software and communication stack**. This workflow integrates an **iterative refinement loop** that enables early validation and iterative improvement. Implementation details such as h_i determination strategies and formal deviation detection thresholds remain subjects for experimental validation.

While proposing significant conceptual and architectural advancements, a primary limitation is the absence of new experimental validation specifically quantifying the benefits of variable discretization and demonstrating the effectiveness of the iterative refinement loop. Future work will focus on empirical validation, quantifying the benefits of variable discretization and demonstrating how detailed software and communication stack modeling in Frost reveals issues overlooked by traditional physical-only simulations, alongside exploring runtime-reschedulable reactors and automating transformations from existing synthesis tool outputs.

References

[1] R. Ruiz and J. A. Vázquez-Rodríguez, "The hybrid flow shop scheduling problem," *European Journal of Operational Research*, vol. 205, no. 1, pp. 1–18, 2010.

[2] M. Balszun, C. Hobbs, E. Fraccaroli, D. Roy, F. Fummi, and S. Chakraborty, "Process Dynamics-Aware Flexible Manufacturing for Industry 4.0," in *18th International Conference on Automation Science and Engineering*. IEEE, 2022, pp. 2375–2382.

[3] M. Balszun, C. Hobbs, E. Fraccaroli, D. Roy, and S. Chakraborty, "Exploiting Process Dynamics in Multi-Stage Schedule Optimization for Flexible Manufacturing," in *27th International Conference on Emerging Technologies and Factory Automation (ETFA)*. IEEE, 2022, pp. 1–8.

[4] T. Blochwitz, M. Otter, M. Arnold, C. Bausch, C. Clauß, H. Elmqvist, A. Junghanns, J. Mauss, M. Monteiro, T. Neidhold *et al.*, "The functional mockup interface for tool independent exchange of simulation models," in *8th international Modelica conference*. Linköping University Press, 2011, pp. 105–114.

[5] M. W. Grieves, "Digital twins: past, present, and future," in *The digital twin*. Springer, 2023, pp. 97–121.

[6] M. Lohstroh, C. Menard, S. Bateni, and E. A. Lee, "Toward a Lingua Franca for Deterministic Concurrent Systems," *ACM Transactions on Embedded Computing Systems*, vol. 20, no. 4, pp. 1–27, 2021.

[7] M. Lohstroh, Í. Í. Romeo, A. Goens, P. Derler, J. Castrillon, E. A. Lee, and A. Sangiovanni-Vincentelli, *Reactors: A deterministic model for composable reactive systems*. Springer International Publishing, 2020, pp. 59–85.

[8] M. Lohstroh, "Reactors: A Deterministic Model of Concurrent Computation for Reactive Systems," Ph.D. dissertation, EECS Department, University of California, Berkeley, Dec 2020. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2020/EECS-2020-235.html

[9] C. Menard, M. Lohstroh, S. Bateni, M. Chorlian, A. Deng, P. Donovan, C. Fournier, S. Lin, F. Suchert, T. Tanneberger, H. Kim, J. Castrillon, and E. A. Lee, "High-performance Deterministic Concurrency Using Lingua Franca," *ACM Trans. Archit. Code Optim.*, vol. 20, no. 4, 2023.

[10] University of Verona. (2024) Glacier Project. (Accessed on 2025/05/19). [Online]. Available: https://esd-univr.github.io/glacier-website

[11] ——. (2025) ICE Laboratory. (Accessed on 2025/08/12). [Online]. Available: https://www.icelab.di.univr.it/?lang=en