# Probabilistic Update Scheduling for Digital Twins: A Semi-Markov Approach

Mikkel Schmidt Andersen
Department of Electrical and Computer Engineering
Aarhus University & Fibo Intercon A/S
Aarhus, Denmark & Videbæk, Denmark
Email: msa@ece.au.dk & msa@fibointercon.com

Cláudio Gomes, Sophia H. Thompson, Peter Gorm Larsen
Department of Electrical and Computer Engineering
Aarhus University
Aarhus, Denmark
Email: {claudio.gomes, sthompson, pgl}@ece.au.dk

***Abstract*—Digital Twins (DTs) often require maintenance throughout their life cycles, as their Physical Twin (PT) counterparts undergo maintenance and evolution. This necessitates software updates to the DT, but when should these updates be done? Updating the DT at the wrong time can lead to inconsistencies between the DT and PT, as well as failures and increased downtime.**

**This study investigates the practicality and usability of applying Semi-Markov Processes (SMPs) to represent the connections and state transitions of the DT-enabled system. SMPs can be used to calculate the probability that all components in the system are collectively in a *safe* state, that is, in a state where updating the DT would result in minimal disruption to the DT-enabled system's operation.**

**We also discuss the limitations of the approach and the future work required to make it robust. Lastly, we present how SMPs can be used for an industrial concrete mixer as our case study, to remove the need for fixed maintenance intervals, which are costly.**

***Index Terms*—digital twins, semi-markov update scheduling, concrete batch plant, simulation, physical twin, cyber-physical systems, process modeling, stochastic scheduling.**

## I. INTRODUCTION

Digital twins (DTs) and physical twins (PTs) have gained attention in recent years, particularly in the context of Industry 4.0 and the Internet of Things (IoT), to improve efficiency, reliability, and safety in a wide range of applications such as smart factories, concrete mixers, and autonomous vehicles. To use this technology, these systems often implement bidirectional communication [1], [2].

Research on DTs has been focused on their design and initial implementation across various application domains. However, few studies have looked into what happens after a DT is deployed [3]. Questions such as how a DT service is delivered to production and how it is maintained, updated, and improved during continuous nonstop operation remain largely open research areas. An important requirement of DTs is that they should be consistent with the PT over its lifetime, which means updating the DT to match the PT as it undergoes maintenance, ages, or any other evolution. As we demonstrate later in this work, the more complex the DT-enabled system (PT + DT) becomes, the more challenging it is to update the DT without causing disruptions to the PT.

Studies have mentioned different approaches for updating models, such as iterative model-update techniques, to keep the DT accurate as the PT ages due to wear and tear [4] and more recent work in the context of distribution networks in the energy sector explored how real-time update algorithms can be used to evolve a DT model [5]. Besides this, the concept of model updates has also been looked at in the research area on dynamic software updating (DSU), where the desired results are the possibility to update software at runtime by updating individual components of the system without necessitating a complete system shutdown [6]. However, the implementation of DSUs is not without its challenges [7]. Particularly, services that maintain an internal state representation, which are dependent on the previous history of inputs (stateful services), can prove challenging to update dynamically without transferring the state.

*Problem.* We hypothesize that DT-enabled systems have safe states in which updates can be carried out with minimal disruption. This highlights the following challenges: (1) recognizing that a DT-enabled system is in a safe state, (2) determining how long that safe state lasts, and (3) predicting when the DT-enabled system will next be in a safe state, so that updates can be scheduled. Traditional approaches rely on gating mechanisms that use static rules or other conditions that assume complete knowledge of the system, such as the software build to be successful, or carry out updates during times when the system's usage is minimal. For DT-enabled systems where users are interacting with it nonstop shows that such rules are insufficient. Here, safe states are not generally known in advance, exhibiting instead stochastic behavior that can be captured by probabilistic models. The results from other application areas, such as process mining and reliability modeling [8], [9], [10], suggest that Markov-chain-like for-

malisms can be used for modeling DT services. However, they have not yet been applied to schedule the updates of DT services intelligently. The need for better support in this area is reflected in recent interviews with DT practitioners, which highlighted issues with tool support, update cycles, and long-term maintainability [11].

*Contribution.* In this study, we present a novel approach to determine when it is probabilistically safe to deploy software updates in DT-enabled systems. We leverage semi-Markov processes (SMPs) to capture the probabilistic behavior of DT services over time. We then demonstrate the effectiveness of our approach through a concrete batch plant case study for cement mixing. The resulting SMP-based method partially addresses the challenges enumerated above.

## II. Running Example: The *Fibo Collect* Concrete Batch Plant

As a running example, let us consider a concrete batch plant, which is called "fibo Collect", illustrated in fig. 1. The plant produces ready-mix concrete by dosing and blending four main ingredients: cement, water, stones, and chemical admixtures. The machine operates by following a specified recipe, which outlines the process for mixing the concrete. The fibo Collect system is highly automated, and its main functions are:

1) **Weighing:** Load cells measure each ingredient.
2) **Mixing:** A pan mixer ensures that the concrete is mixed well.
3) **Discharge:** The fresh concrete is released into a waiting truck for transport within 5 minutes.


Fig. 1. fibo Collect – concrete batch plant – Fibo Intercon A/S

The goal of the batch plant is to produce high-quality concrete while minimizing waste and ensuring efficient operation. However, while installing and later maintaining these machines, we encountered that they often require continuous updates to improve the efficiency and the quality of the product, as our internal tools evolves. During the busiest month, a machine may be updated multiple times to support new data collection metrics and enable more informed decisions. We want to increase the update frequency for the machines, without coordinating when the best time to do it is, and risking downtime for customers' machines. We can also add that an update cannot simply be done at night due to the possibility that something might go wrong at the physical plants and would thus require a person to handle and fix the issues. Using on-call night workers is expensive, and we would like to minimize this possibility by determining the most efficient time, where we have the lowest probability of an error interfering with the machine.

Software updates to a DT service would require the machine to be taken offline, which might disrupt ongoing operations or render some services unavailable during critical phases, such as mixing. It is important to determine when we can safely go in and update the systems without fixed maintenance intervals, as these often require coordination with different stakeholders and are expensive both in terms of man-hours and in lost revenue from missed customers. If we can give a high probability that the system is in a safe state, so that initiating a software update will not disrupt operations it would be beneficial. By using probability, we are confident that the most optimal time to send an update signal can be calculated, which will ensure the update occurs when the system is in a safe state, not mixing concrete or serving other services. An update launched at the wrong moment can potentially cause issues, such as interrupting the collection of data from sensors, freezing the Human-Machine Interface (HMI), or, in the worst case, causing downtime of the machine, thus breaching SLA agreements.

Due to these potential issues, a typical software update process for these systems necessitates coordination between the owner and the software team. A scheduled time will be allocated in which the machine will be taken offline and then updated. They wish to determine, based on user behavior, when we can update the machine without having to take it offline, thereby lessening the burden on all parties and avoiding the fixed maintenance intervals for the machine.

## III. Background and Problem Formulation

In this section, we provide an overview of Markov chains and SMPs. Our SMPs are introduced later in section IV (e.g., see fig. 2).

A Markov chain is a class of discrete stochastic processes (i.e., a sequence of random variables $\{X_n\}$ with $n \in \mathbb{N}$) over a discrete state space $S$ that satisfies the Markov property: the probability of a future state depends only on the current state, and not any previous states Mitrani [12]. If we let $n \in \mathbb{N}$ and let $j, i_1, \ldots, i_n \in S$, this property can be mathematically expressed as:

$$\begin{aligned} &P(X_{n+1} = j | X_n = i_n, X_{n-1} = i_{n-1}, \ldots X_1 = 1) \\ &= P(X_{n+1} = j | X_n = i_n) \end{aligned} \tag{1}$$

where $X_n$ is the state of the process after $n$ transitions, and $P(X_{n+1} = j|X_n = i_n)$ is the probability of transitioning to state $j$ after the next transition, given that the process is in state $i_n$ after $n$ transitions. In a Markov chain, the evolution of the system is described by $Q_{ij}(n)$, which is the probability that the chain will move from state $i$ to state $j$, after $n + 1$ transitions. Relating to the above equation, we have:

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

as described by Mitrani.

Markov chains have been applied in various fields, such as process mining for smart homes [9], maintenance of gas analytical systems [8], degradation-based reliability [13], and price returns for trading stocks [14]. One limitation of Markov chains, however, is that they have no inherent representation of time spent in a state. To represent time, one would have to create artificial states to emulate it. Crucially, if one has a probability distribution for the time spent in a state, it is cumbersome to create a Markov chain that captures this distribution.

SMPs are an extension of the Markov chains that address this limitation. They allow for a wait time (alternatively called holding time or sojourn time) in a state before transitioning to the next, which makes them better suited to model a DT service based on observed behavior patterns. Informally, in SMPs, when we are in state $i$ and want to transition to a new state $j$, we do so by analyzing the state transition probabilities and finding the next state. After choosing the next state $j$, we then spend a random amount of time in state $i$ (the wait time) before transitioning to state $j$. The waiting time can depend on the current state and the chosen next state. An introduction to the formalism, along with more details, is presented in [15].

An SMP consists of: a set of states, a set of transition probabilities, and a set of wait time distributions. Compared to Markov chains, SMPs allow the system to remain in a state for a specific period before transitioning to the next state. The duration and the next state are probabilistically related. For instance, given a current and next state, the duration can follow a uniform distribution or an exponential distribution. The following equations formally define SMPs, adapted from Medhi [16]. The transition function, also called the renewal matrix, is expressed as eq. (2). In addition, the wait time distribution is defined as:

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

where:

- $X_n$ is the state happening at the $n$-th transition,
- $X_n$ is a Markov chain,
- $T_n$ is the time of the $n$-th transition,
- $T_{n+1} - T_n$ represent the wait time, in state $i$ before transitioning to state $j$ and
- $W_{ij}(t)$ is the probability that, the wait time in state $i$ before transitioning to $j$ is less than or equal to $t$, given the process is in state $i$ at transition $n$ and the next state is $j$.

Algorithm 1 shows the pseudocode for simulating an SMP.

**Algorithm 1** Semi-Markov Process Simulation Algorithm

**Require:** Simulation time $T$, initial state $i$

$t \leftarrow 0$
$state \leftarrow i$
**while** $t < T$ **do**
    Retrieve transition set for current $state\ i$
    Extract transition probabilities
    Sample $next\_state\ j$
    Identify selected transition $i \rightarrow j$
    Sample wait time $w$ from selected transition $j$
    Save $(t,\ t + h,\ state)$ to the path
    Determine new time = $t \leftarrow t + h$
    Determine new state $state \leftarrow next\_state$

## IV. Update Scheduling with Semi-Markov Processes

In this section, we describe the approach to update scheduling using SMPs and how we implemented a prototype tool to simulate the set of SMPs, with a focus on the fibo Collect system. The tools and techniques used to combine and simulate SMPs will be described, along with the assumptions made about the system.

The approach follows the idea that a DT and its associated PT can be modeled as a set of SMPs, where each SMP represents a distinct part of the system. In our example, as shown in fig. 2, three SMPs were developed, where each one represented one part of the DT-enabled system. In this case, the PT is represented as a single entity called the Physical Twin. Whereas the DT contains two SMPs, the 'Anomaly Detector' and the 'Flow Rate Detector'. The key aspect is to model the software components in the system that are critical to its operations. Each SMP contains a set of transitions with their associated probability of transitioning (transition probability) and a function describing the wait time, which is the time we wait before transitioning to the next state.

For our models, we had the following assumptions:

- That each system can be modeled as an SMP with accurate transition probabilities and wait times.
- That our wait time is independent of the transition probability.

Regarding the second assumption, it is not a requirement that the SMP are independent of each other. However, it does reduce complexity when simulating the processes together, as the more dependent a service is on another, the lower the probability is that a service can discover a period in which it can safely update within the specified time.

### A. Prototype Implementation

The tool, which simulates the SMPs, is developed using Python version 3.13. The procedure for using the tool involves creating an SMP that describes the system. This can be done either manually, if there is sufficient system knowledge, or as an automated process by reviewing system logs and historical data. In our case, the SMPs were developed based on system domain knowledge, as shown in fig. 2. Each of the SMPs

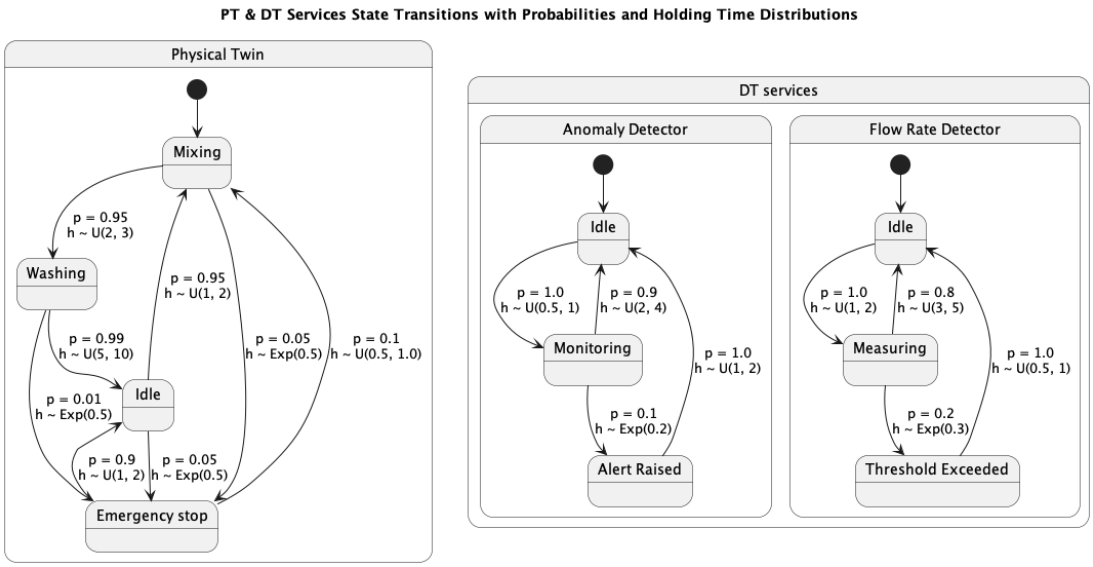


Fig. 2. Example of a set of SMPs for the fibo Collect system, and two mock DT services. p indicates the probability of transitioning to that state, and h is the wait time distribution, which indicates the time spent in a state before the transition happens. The wait time is independent of the transition probability.

contains the associated transition probabilities, as well as the wait time distributions for the specific state transition. The tool uses this as input to simulate the set of SMPs simultaneously. Below is the description of the input and the algorithm used to simulate the SMPs, which is the *safety specification* of the system.

- `SMPs` A list of SMPs.
- `Safe states` A dictionary containing the SMP name and the list of safe states for that SMP.

Besides this, a required input for each of the SMP must contain the following information:

- `name`: The name of the SMP.
- `States`: The states represented as a dictionary, with the name of the state, and the probability and wait time distributions.

The tool utilizes the required inputs and simulates all the SMPs over a specified period, verifying whether the system is in a safe state overall. It does this by correlating each SMP against the specified safety specification. The simulation follows algorithm 1 and aligns with the mathematical equations described in section III.

The algorithm is as follows: To simulate the SMPs, we can split them into three categories: (1) We select the transition randomly, based on the available states in the SMP. (2) We then sample the waiting time, from the distribution function, and await that time before transitioning to the next state and (3) We apply Monte Carlo, as we simulate outcomes of the SMPs over a given large number of simulations $N$ (e.g., $N = 1500$) simulations, and then calculate the probability of it being safe, by estimating the quantities from the simulations.

The main algorithm used to perform the simulation is shown in listing 1. It is inspired by the procedure described in [16], and implements the semi-Markov elements as defined in eq. (3).

**Sample Size Determination.** We briefly sketch how to determine $N$. There are two approaches: **A)** $N$ is chosen based on the convergence of a random variable distribution of interest; **B)** $N$ is chosen based on the desired level of confidence and precision. Approach 1 entails stopping new simulations when, for instance, the maximum duration of a safe state (as in fig. 4) stabilizes. Approach B works by formulating a Bernoulli process ($X_i$), whose possible outcomes are success or failure (i.e., $X_i = 1$ or 0). Then, $X = \sum_i^N X_i \sim \text{Binomial}(n, p)$, and approach B focuses on finding an $N$ such that $\Pr\left(|\hat{p} - p| \leq E\right) \approx 1 - \alpha$, where $\hat{p}$ is estimated from the simulations, $E$ is the precision, and $1 - \alpha$ is the confidence level. We then determine $N$ by applying the Central Limit Theorem and using the properties of the normal distribution.

```
while t < simulation_time:
    # Get the current object transitions
    state_obj = chain.states[state]
    transitions = state_obj.transitions

    # (Q_ij) Extract distribution for possible next states
    probs = [tr.probability for tr in transitions]

    # Choose next state X_{n+1} based on Q_ij
    next_state = np.random.choice([tr.to_state for tr in
     transitions], p=probs)

    # Time until next state (i -> j) wait timt W_{ij}(t)
    selected_transition = next(tr for tr in transitions if
     tr.to_state == next_state)
    waiting_time = selected_transition.waiting_time()

    # Enter time, leaving time, state
    path.append((t, t + waiting_time, state))

    # Move time forward and update our current state
    t += waiting_time
    state = next_state
```

Listing 1. SMP Simulation Algorithm

### B. Evaluation and Results

We evaluated our SMP based scheduling approach using the set of processes illustrated in fig. 2. Our objective was to assess the tools' ability to identify safe deployment windows and characterize the distribution of safe and unsafe system states during update procedures, such that an update would not be performed at the wrong time. In fig. 3, we show the concept of safe deployment windows within a typical operational timeline, as a plot of states. The tool predicts these windows by simulating the system's state evolution over 30 minutes, taking into account the wait time distributions and transition probabilities defined in the SMPs. Operators can use these predictions to plan updates during periods with a high probability of safety, minimizing the risk of disruption. To validate the SMPs, we ran simulations from different initial

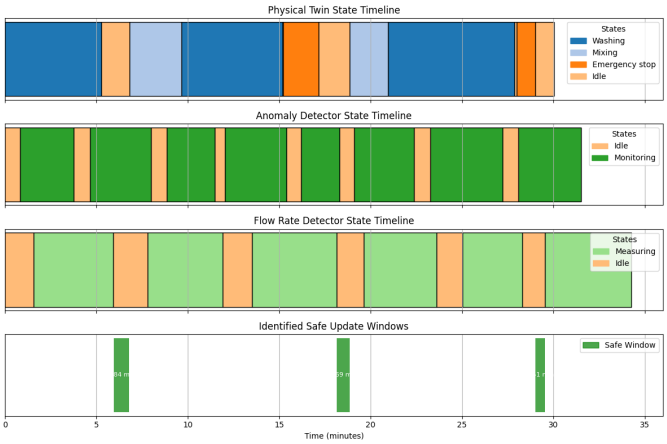


Fig. 3. Diagram illustrating the safe deployment window concept. The simulated system can be seen in fig. 2 and consists of three processes, where the safe states are *idle* and *monitoring*. This is one simulation of the system.

system configurations. We can observe that when starting from a *safe* system state, the results shown in fig. 4 indicate that the system remains in a safe configuration for significant periods, with multiple windows available for update deployment.

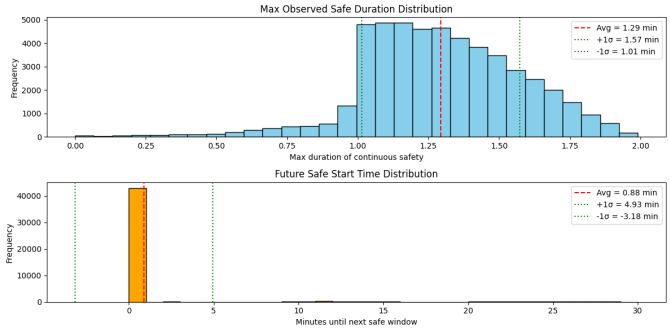


Fig. 4. Safe state durations based on 1500 simulations. Most runs estimate a high probability of the system remaining safe for over 1 minute—the target threshold.

The histogram in the figure shows the duration of the simulated safe states, highlighting that safe windows are sufficiently frequent for the running example. This supports the tool's applicability in real-time or near-real-time industrial settings. However, further tests are needed on a real system to validate its applicability in a real-world setting.

We also experimented with system configurations where the system started from an *unsafe* initial state. It can be seen on fig. 5 that the system has a high spread, making it difficult to determine when a safe update period should be conducted. However, it can still be used as a reasonable estimate for when the next update window period is likely to occur. Overall,

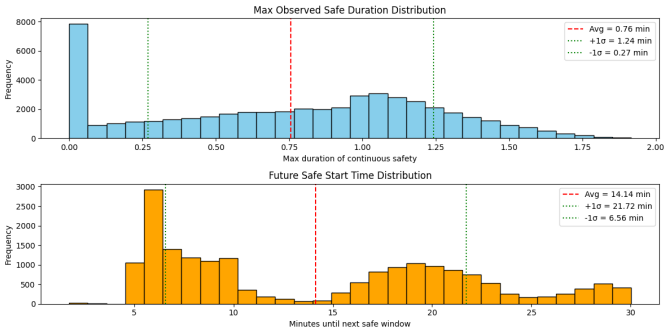


Fig. 5. Safe state durations based on 1500 simulations. Most runs estimate a low probability of the system remaining safe for over 1 minute—the target threshold.

the results indicate that SMPs can help operators by applying a data-driven approach to update scheduling in DTs, thereby reducing the risk of unintended service disruptions for systems that can run nonstop, while still allowing for small windows of opportunity for high-probability deployment without risking downtime.

### C. DISCUSSION

We will discuss the potential of using SMPs to support safe update scheduling in DTs. However, our current findings are based solely on simulation studies. Future work will trial the approach on the running example that inspired us to model the DT services.

The set of SMPs evaluated in this paper was designed, to the best of our ability, to match the expected behavior of the real system. However, it has not yet been validated against the case study, nor was it created from historical state logs. Although the results provide insights into the feasibility and potential benefits, several important considerations and limitations must be addressed before the approach can be deployed in a live industrial setting. We will therefore reflect on the implications of our simulation results, discuss the representativeness of our models, and identify the limitations and practical challenges associated with them.

The simulation results can be interpreted in two ways. As concluded in the earlier section, fig. 4 shows that when the initial state is *safe*, there is a high probability that the system will remain safe over time, provided the wait time distribution allows sufficient time before transitioning. However, when the state is *unsafe*, as shown in fig. 5, it indicates that the timing of the following safe period is almost random and can be

difficult to derive any value from. Consequently, our tooling should estimate an approximate probability for the next safe state window, allowing an update to be scheduled accordingly. However, this can be challenging based on current results.

As mentioned above, The SMPs were not derived from historical data, but were constructed based on our understanding of the PT, and two mock DT services. This limitation currently restricts our ability to conduct a practical test, which is why future work prioritizes extending the tooling to generate SMPs directly from data stored in a time-series database. Besides this, the formalism of SMP allows it to be used for more complex systems, however for our needs the attribution of how the waiting time, affects the transition probability matrix was relaxed such that, we are allowed to sample the next state, and then determine the wait time needed in the current state before transitioning.

## V. Related Work

The management and orchestration of DTs in cyber-physical systems (CPS) have received increased attention in recent years, particularly in scenarios that require runtime verification, adaptation, and safe software updates.

Kamburjan et al. [17] propose an automated method for declarative lifecycle management in DTs to address the shifts between the different lifecycle stages for the PT. These methods utilize descriptions of the lifecycle of components and their associated DT parts, leveraging knowledge graphs and ontologies. The authors mentioned that reflecting the lifecycle evolution is challenging. In addition, Aissat et al. [18] mentioned that the underlying complex systems of a DT must be continuously updated to meet user requirements and to enable the system to improve over time. They presented the tool Juno-OPS, which is a DevOps framework to support the built assets from DTs and can help engineers without DevOps knowledge to leverage best practices and deploy faster to production. However, they did not mention how to ensure that a service is ready to be taken offline for updates, but they implemented rigorous testing and quality assurance techniques in their pipelines. SMPs can complement this by introducing a probability check between the last layer, right before it is deployed in the production environment. In addition to these studies, SMPs have been applied in various fields, such as [9], which utilizes semi-Markov models for process mining in smart homes to detect anomalies. The results indicate that this approach fits well with smart home data. The work by [8] focused on developing a semi-Markov model of a complex gas analytical system, enabling them to create an optimal maintenance strategy to reduce the cost of maintenance for the airspace industry, which further implies that SMPs are well-suited for update scheduling in DTs.

## VI. Conclusion and Future Work

This paper presents our current work on developing a tool to determine the optimal time for deploying updates to either DT services or the PT software. Our evaluation revealed that as the number of modeled states increased, the time available for safe updates decreased, resulting in a reduction in the number of safe update windows. The starting states from the different SMPs also significantly impacted the available safe update windows, indicating that the more SMPs there are to simulate, the harder it is to obtain a safe update window. This challenge can be addressed by decomposing the SMP into sub-SMPs, allowing each to focus on the components that matter most for specific updates. In terms of tool development, the current prototype supports the manual specification of SMPs and therefore relies on the user to define the safe states and model the DT and PT as SMPs. The evaluation demonstrated that this approach can be effective in reducing the risk of a disruptive update. However, several limitations remain. Additionally, the accuracy of the update recommendations is directly dependent on the quality of the SMPs, especially the accuracy of the wait time distributions and transition probabilities.

Future work will focus on further developing the tool to generate SMPs from historical state logs automatically by applying state-of-the-art techniques, such as [19] to discover SMPs from data. We also aim to implement a real-time monitoring component that uses the generated SMPs to gate and apply updates at optimal times, based on probabilistic safety estimates. Here it is important to stress that a state is considered *safe* when it satisfies the safety specification defined for the system, as mentioned in section Section IV-A. The probability that we estimate is the likelihood of *remaining safe* for a target duration $n$. For example, an update may only be scheduled if there is at least a 95% probability of the system staying safe for the next 5 minutes. Finally, we aim to integrate the complete tool chain in the case study.


## Acknowledgment

This work was supported by the Innovation Fund Denmark under the project *Digital Twin Deployment for Enhanced Autonomy and Scalability of Ready-Mix Concrete Batch Plants*


## References


[1] J. Fitzgerald, C. Gomes, and P. G. Larsen, *The Engineering of Digital Twins*. Springer, 2024.

[2] M. Javaid, A. Haleem, and R. Suman, "Digital twin applications toward industry 4.0: A review," *Cognitive Robotics*, vol. 3, pp. 71–92, 2023.

[3] B. Zhang, G. Ding, Q. Zheng, K. Zhang, and S. Qin, "Iterative updating of digital twin for equipment: Progress, challenges, and trends," 10 2024.

[4] W. Kim, S. Kim, J. Jeong, H. Kim, H. Lee, and B. D. Youn, "Digital twin approach for on-load tap changers using data-driven dynamic model updating and optimization-based operating condition estimation," *Mechanical Systems and Signal Processing*, vol. 181, p. 109471, 2022.

[5] J. Shen, L. Hu, Y. Yang, Y. Li, and P. Lou, "Real-time update algorithms for digital twin models of distribution network equipment under internet of things and optical imaging technology," *Scientific reports*, vol. 15, p. 5910, 12 2025.

[6] M. Wahler, S. Richter, and M. Oriol, "Dynamic software updates for real-time systems," in *Proceedings of the 2nd International Workshop on Hot Topics in Software Upgrades*. Orlando Florida: ACM, Oct. 2009, pp. 1–6. [Online]. Available: https://dl.acm.org/doi/10.1145/1656437.1656440

[7] D. Mlinarić, "Challenges in Dynamic Software Updating," vol. 9, no. 1. [Online]. Available: https://doi.org/10.18421/TEM91-17

[8] V. Bobkov, O. Kanishchev, and I. Men'Shova, "The semi-markov model of operation and maintenance of gas analytical system," in *Journal of Physics: Conference Series*, vol. 1925. IOP Publishing Ltd, 6 2021.

[9] S. McClean and L. Yang, "Semi-markov models for process mining in smart homes," *Mathematics*, vol. 11, 12 2023.
[10] A. Świderski, A. Borucka, M. Grzelak, and L. Gil, "Evaluation of machinery readiness using semi-markov processes," *Applied Sciences (Switzerland)*, vol. 10, 2 2020.
[11] H. M. Muctadir, D. A. Manrique Negrin, R. Gunasekaran, L. Cleophas, M. van den Brand, and B. R. Haverkort, "Current trends in digital twin development, maintenance, and operation: an interview study," *Software and Systems Modeling*, 10 2024.
[12] I. Mitrani, *Probabilistic modelling*. USA: Cambridge University Press, 1998.
[13] J. P. Kharoufeh, C. J. Solo, and M. Y. Ulukus, "Semi-markov models for degradation-based reliability," *IIE Transactions*, vol. 42, no. 8, pp. 599–612, 2010.
[14] G. D'Amico and F. Petroni, "A semi-markov model for price returns," *Physica A: Statistical Mechanics and its applications*, vol. 391, no. 20, pp. 4867–4876, 2012.
[15] S. Ross, "- renewal theory and its applications," in *Introduction to Probability Models (Eleventh Edition)*, eleventh edition ed., S. Ross, Ed. Boston: Academic Press, 2014, pp. 409–479. [Online]. Available: https://www.sciencedirect.com/science/article/pii/B9780124079489000074
[16] J. Medhi, *Stochastic processes*. New Academic Science Limited, 2012.
[17] E. Kamburjan, N. Bencomo, S. L. Tapia Tarifa, and E. B. Johnsen, "Declarative lifecycle management in digital twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 353–363.
[18] S. Aissat, J. Beaulieu, F. Bordeleau, J. Gascon-Samson, E. A. Poirier, and A. Motamedi, "Juno-ops: A devops framework for the engineering of digital twins for built assets," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, 2024, pp. 496–506.
[19] A. Kalenkova, L. Mitchell, and M. Roughan, "Performance analysis: discovering semi-markov models from event logs," *IEEE Access*, 2025.