

Asymmetric Timed Refinement for Correct Post-Deployment Updates in Real-Time Systems

Luca Brodo[✉], Giuseppe Scalora[✉], Stefan Henkler[✉]

Hamm-Lippstadt

University of Applied Sciences

Lippstadt, Germany

{name.last_name}@hshl.de

Abstract—Model-based systems engineering is a well-established framework for developing post-deployment updates in cyber-physical systems. In real-time and safety-critical settings in particular, these updates must be shown to preserve previously verified properties. Refinement relations over timed models provide an attractive alternative to full model checking, as they establish correctness by behavioral inclusion and can be checked efficiently and compositionally. However, traditional relations enforce symmetric timing preservation for all observable events, rejecting adaptations such as buffering or asynchronous communication, which preserve all safety-critical output deadlines but would require relaxed reception timing. We address this limitation by introducing Relaxed Weak Timed Bisimulation (RWTBS), a novel refinement relation that strictly preserves output timing while permitting controlled relaxation of input reception timing. We prove key property preservation and present a symbolic algorithm to decide RWTBS. Our evaluation demonstrates that RWTBS accepts updates rejected by classical refinement, can be used to correctly identify the operating boundary beyond which updates become unsafe, and achieves speedups of up to five orders of magnitude over model-checking with memory savings of three orders of magnitude.

Index Terms—Refinement, Model-Based Design, Post-Deployment Updates

I. INTRODUCTION

Model-based design workflows have become the standard mechanism for developing and validating post-deployment component updates in cyber-physical systems [1], [2], [3], where evolving requirements [4], changing operational conditions and optimizations necessitate continuous evolution. These workflows abstract the behavior of the deployed system in virtual or simulation-based models, enabling engineers to design and validate updates offline prior to deployment.

In safety- and time-critical systems in particular, any proposed update must preserve previously verified guarantees, including functional correctness, real-time constraints, and contractual obligations toward the environment. When updates are derived from model-based workflows, a formal validation mechanism is required to bridge the gap between the virtual model and the deployed system, ensuring correctness while avoiding the computationally prohibitive cost of model-checking the entire system after each update.

Refinement relations over timed execution models provide an attractive solution. They establish behavioral inclusion between successive system versions, ensuring that updates cannot introduce new observable behaviors that violate previously

verified properties or exceed platform constraints. Because refinement can be checked compositionally, thus validating each component in isolation while preserving system-level guarantees, and often decided by efficient symbolic algorithms, it avoids the computational cost and the state-space explosion associated with model checking.

Traditional refinement relations, such as timed bisimulation and its variants [5], [6], [7], however, rely on idealized symmetric timing assumptions: they require strict timing correspondence for all observable events, including both sent and received actions. This symmetry is unnecessarily restrictive in practice. From a contractual perspective, the timing of emitted events is a real-time obligation and must be preserved, whereas the timing of received events reflects responsiveness to stimuli and can be relaxed as long as external guarantees are not violated. Rejecting this timing asymmetry rules out realistic forms of evolution, such as buffering or asynchronous communication, which could have great optimization potential and thus limits the applicability of classical refinement relations in model-based incremental verification.

To address this limitation, this paper introduces Relaxed Weak Timed Bisimulation (RWTBS), a refinement relation that accepts this asymmetry in timing preservation, thereby able to accommodate realistic adaptations requiring relaxed input timing. We show RWTBS preserves essential system properties across updates, including output deadlines and compositional contract preservation, and present a zone-based symbolic refinement-checking algorithm. The role of RWTBS in post-deployment validation is clarified through a multi-view evolution framework that structures the relationship between deployed systems, candidate updates, and system specifications.

The paper is organized as follows. Section II introduces a motivating example. Section III discusses related work. Section IV presents the multi-view evolution framework. Section V establishes the formal foundations. Section VI defines RWTBS, Section VII presents a symbolic algorithm, and Section VIII establishes the properties preserved by RWTBS. Section IX evaluates the approach, and Section X concludes.

II. MOTIVATING EXAMPLE: POST-DEPLOYMENT EVOLUTION OF A DRONE FLEET

Consider a monitoring drone deployed for sugar beet field survey, equipped with a GPS receiver and an inertial mea-

surement unit (IMU) interfaced to a flight controller. The control component acquires GPS fixes and emits corrected waypoint commands, which must be issued within 100 ms of fix acquisition to maintain trajectory accuracy.

In the originally deployed system, position control is driven purely by GPS, as the on board CPU has insufficient computational resources to run both the attitude controller and a real-time GPS-IMU fusion filter within 100 ms. Without ego-motion compensation, a fix must be consumed immediately, to avoid drift due to the drone's movement.

After deployment, optimization of the attitude-control loop freed enough CPU capacity to run a GPS-IMU sensor fusion algorithm. Using IMU motion estimates, the system can compensate for the drone's movement and keep a GPS fix usable for up to 50 ms after acquisition. This gives the scheduler flexibility to delay fix processing slightly and align it with the regular flight-control cycle, reducing unnecessary CPU wake-ups and context switches.

The 100 ms deadline remains unchanged. Classical timed refinement rejects this update for violating strict timing correspondence on the GPS-fix input event, even though the output deadline is preserved. This motivates a refinement relation that captures the natural asymmetry: strict preservation of output timing, with controlled relaxation of input reception timing.

III. RELATED WORK

Iterative design and incremental evolution are standard practices in cyber-physical and real-time systems development, where systems are refined across multiple design cycles to accommodate changing requirements, environments, or operating conditions. To manage verification complexity, analysis is typically performed compositionally or partially, focusing on modified components rather than re-verifying the entire system [8], [9]. While essential for scalability, such practices implicitly assume that changes preserve the semantic and timing assumptions under which earlier verification results were established. In practice, however, refinements often introduce additional processing stages, buffering, or asynchronous execution, which may silently invalidate these assumptions without being exposed by partial re-verification.

The resulting divergence between abstract verification models and the actual execution, known as semantic drift [10], is a well-recognized challenge to dependable system evolution. In the self-adaptive context, the works of [11] explicitly argue that runtime models must co-evolve with the system to remain meaningful abstractions, while also acknowledging the challenge of preserving behavioral semantics during adaptation. Existing mitigation strategies primarily rely on runtime verification [12], structural traceability [13], [14], or reactive loops [15]. While effective for post-deployment fault detection, these approaches do not formally constrain the adaptation process itself, allowing inconsistencies to accumulate across design cycles before being detected.

RWTBS aligns with the concept of Interface Automata [16] and the Timed I/O Automata (TIOA) specification theory [17], which distinguish between actions controlled by the system and the environment. These frameworks define refine-

ment through input-enlargement and output-reduction, ensuring functional substitutability. The ECDAR tool [18], based on TIOA, provides compositional refinement for timed systems with I/O distinction. TIOA refinement treats inputs and outputs asymmetrically with respect to action enabledness: the refined specification must accept all inputs of the abstract one and may only produce outputs already permitted. However, this asymmetry does not extend to timing: the delay-matching condition is symmetric for all observable actions regardless of polarity [17], and TIOA's input-enabledness semantics requires inputs to be consumed immediately upon arrival, leaving no room to distinguish between event arrival and event processing. Consequently, a component that buffers an input and consumes it later violates delay matching even if all output deadlines remain satisfied and this cannot be remedied by relaxing the delay condition alone, since the buffered arrival/consumption split has no representation in the TIOA semantic model. As a result, a component that defers input consumption through buffering cannot be validated by ECDAR, leaving engineers without a compositional verification path for a practically common class of updates. RWTBS addresses this by grounding the model in explicitly buffered input ports, introducing I/O asymmetry at the level of timing itself.

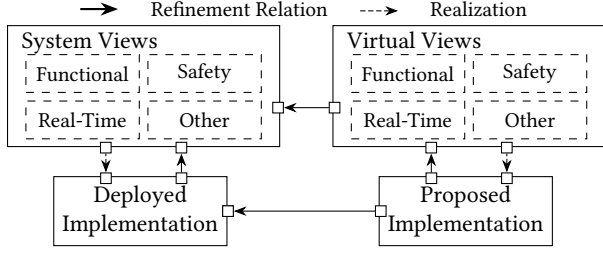
To guarantee correctness before deployment, iterative evolution must be treated as a refinement problem. Refinement is a well known concept both at the model-level [9] as well as the property level [19], [20]. However, classical timed refinement frameworks and faster-than preorders [5], [6], [7] rely on idealized, symmetric timing assumptions. They require strict timing correspondence for all observable actions, rejecting realistic execution-level adaptations. Similarly, robust semantics for timed automata [21], [22] tolerate timing perturbations uniformly across all clocks or delays, lacking the I/O-aware asymmetry needed to preserve strict output contracts while relaxing input reception.

The engineering necessity of asymmetric timing is well-established in hardware and control. Latency-insensitive design [23] and elastic circuits [24] decouple computation from communication to tolerate variable reception latency using buffers. Likewise, assume-guarantee and game-based models internalize environment-induced delay to synthesize robust controllers [25], [26]. RWTBS complements these architectural and synthesis-based approaches by formalizing this I/O asymmetry directly at the semantic refinement level. By doing so, RWTBS accommodates realistic execution evolution while strictly preserving externally observable timing guarantees, enabling sound partial re-verification without semantic drift.

IV. SYSTEM EVOLUTION FRAMEWORK

RWTBS is designed to operate in the context of model-based post-deployment evolution, where a deployed implementation and a candidate update derived from a virtual or simulation-based model must be formally related. Figure 1 illustrates this setting. We structure system evolution using the notion of views, inspired by established multi-view design principles [27] and supported by various system architectures [28], [29].

Figure 1: Relation between views



A system view captures a specific concern of the system such as functional behavior, timing or safety requirements through a formal model. Separating concerns into semantically related views enables concern-specific reasoning while maintaining consistency across abstraction levels. Each implementation is required to refine its corresponding system view, ensuring that established guarantees are preserved.

To support system evolution in model-based workflows, we introduce virtual views. These are formal models that serve as the basis for developing and reasoning about proposed optimizations, adaptations, and updates of the deployed system. Each virtual view addresses a subset of the concerns covered by the corresponding system view, and multiple virtual views may therefore correspond to a single system view. Each virtual view must represent a valid refinement of the corresponding system view, ensuring that proposed evolutions adhere to the intended specifications.

Deployment of an update then requires that the proposed implementation refines both its corresponding virtual views and the currently deployed implementation. The former ensures specification-level correctness, while the latter ensures that no new observable behaviors are introduced that violate previously verified properties or contractual obligations toward the environment. It is in particular the second refinement that RWTBS is designed to establish.

V. FORMAL SEMANTICS FOUNDATION FOR SYSTEM EVOLUTION

We establish the concept of time within a domain denoted as *Time*, where the instances of time possess a total order denoted by $\leq$. A segment of the time domain refers to a subset *Time* that is closed downward, meaning that if $t \in T$, then all elements t' such that $t' \leq t$ also belong to T .

To capture interactions, we consider a set of system ports $P \subseteq \mathcal{P}$, where each port is associated with a type that has its own domain of values. Ports serve as the observable interface of a system and its components, governing its interaction with the environment and allowing to distinguish between observable and internal behavior. Ports are classified as being either input ($\mathcal{P}_{in} \subseteq \mathcal{P}$) or output ($\mathcal{P}_{out} \subseteq \mathcal{P}$) ports, depending on whether they receive data from or send data to the environment, respectively. Each input port is equipped with a FIFO buffer. An incoming event is enqueued in the buffer upon arrival and the receiving component dequeues and processes it within its timing constraints. We assume that shared ports between components have compatible polarities (output-input),

preserving the distinction between sent and received events. To prevent an unbounded accumulation of reception delays, we enforce an alternating send/receive constraint. This means a component must send an event before receiving the next one. Under this assumption, the occupancy of reception buffers is bounded by 1, as the next event cannot be received until the previous one has been sent.

We associate each component M with a Timed Transition System (TTS) that serves as its execution model:

Definition 1. A *Timed Transition System (TTS)* is a tuple $\mathcal{T} = (S, s_0, \Sigma_E, \Sigma_D, \rightarrow)$, where:

- S is a set of states, where s_0 is the initial state, and
- Σ_E is a set of *event labels*,
- Σ_D is a set of *delay labels*, with $\delta \in \Sigma_D$ representing the elapse of $\delta \in \text{Time}$ time units.
- $\rightarrow \subseteq S \times (\Sigma_E \cup \Sigma_D) \times S$ is the transition relation.

A timed path from a state s to a state s' of a TTS $\mathcal{T}$ is a finite sequence $s = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n = s'$ where each $a_i \in \Sigma_E \cup \Sigma_D$. The accumulated delay before μ along a path π , written $\sum_{\mu}^{\pi} \delta$, is the sum of all $\delta \in \Sigma_D$ occurring before the first occurrence of μ in π . When the path π is clear from context, we write $\sum_{\mu} \delta$. A trace induced by a timed path $\pi = s_0 \xrightarrow{a_1} s_1 \xrightarrow{a_2} \dots \xrightarrow{a_n} s_n$ is the sequence of labels $\sigma = a_1, a_2, \dots, a_n$. The semantics $\llbracket M \rrbracket$ is the set of all traces induced by its TTS.

A subset of events in Σ_E is associated with the ports of the component, allowing us to distinguish between input and output events. An event label $\mu \in \Sigma_E$ is an output event if it is associated with a port $p \in P_{out}$, and an input event if associated with $p \in P_{in}$. We denote by Σ_{out} and Σ_{in} the sets of output and input event labels, respectively.

To formalize the expected behavior of components, we use temporal logic formulas, such as Timed Computation Tree Logic (TCTL) [30], to define sets of acceptable traces. The semantics of a formula ϕ , denoted by $\llbracket \phi \rrbracket$, is the set of traces that satisfy ϕ . A component M satisfies a formula ϕ , written $M \models \phi$, iff $\llbracket M \rrbracket \subseteq \llbracket \phi \rrbracket$.

Given two components M and M' , we say that M' refines M if $\llbracket M' \rrbracket \subseteq \llbracket M \rrbracket$. This definition ensures that refinement may restrict observable behaviors but cannot introduce new ones. As a result, any property satisfied by M remains valid for M' . Refinement therefore provides a sufficient condition for correctness preservation under system evolution.

To ensure composability and modularity, we employ contract based design, as defined in [8]. A contract $C = (A, G)$ is a pair, where A and G are sets of traces representing, respectively, the assumptions on the environment and the guarantees provided by the component. For guarantees expressing timing constraints on sent events, we assume that they are of the form of upper bounds (deadlines) on the transmission time of output events [31]. This is a common pattern in real-time systems, where output events must be emitted within specified deadlines [32] and these bounds must be preserved under refinement. The semantics of a contract C , denoted by $\llbracket C \rrbracket$, is defined as the set of traces for which the assumptions imply the guarantees, formally $\llbracket C \rrbracket = \{\sigma \mid \sigma \notin A \vee \sigma \in G\}$. A component M satisfies a contract C , written $M \models C$, if

all its observable behaviors are allowed by the contract, i.e. $\llbracket M \rrbracket \subseteq \llbracket C \rrbracket$. A contract $C' = (A', G')$ refines another contract $C = (A, G)$, written $C' \preceq C$, if $A \subseteq A'$ and $G' \subseteq G$. This ensures that any component satisfying C' also satisfies C . Two contracts $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ over components are *compatible* iff $G_1 \cap G_2 \cap (A_1 \cup A_2) \neq \emptyset$. Their *composition* is $C_1 \otimes C_2 = ((A_1 \cap A_2) \cup \overline{G_1 \cap G_2}, G_1 \cap G_2)$. Composition is sound [8]: if $M_1 \models C_1$ and $M_2 \models C_2$, then $M_1 \parallel M_2 \models C_1 \otimes C_2$.

In our framework, contracts are defined over the interface ports of a component. A contract constrains only the externally observable input/output behavior, abstracting from internal execution structure, ensuring that refinement can be reasoned about solely in terms of observable behavior.

We make use of *Timed Automata (TA)* [33], [34] as a syntactic representation of the component models. Formally, a TA is defined as follows.

Definition 2. A TA is a tuple $(L, \ell_0, C, \Sigma, E, \text{Inv})$, where:

- L is a finite set of locations with initial location $\ell_0 \in L$,
- C is a finite set of clocks,
- $\Sigma = \Sigma_E \cup \Sigma_D$ is the action alphabet, partitioned into event-driven and delay-driven actions with $\Sigma_E \cap \Sigma_D = \emptyset$,
- $E \subseteq L \times \mathcal{B}(C) \times \Sigma \times 2^C \times L$ is a finite set of edges, where $\mathcal{B}(C)$ denotes clock constraints,
- $\text{Inv} : L \rightarrow \mathcal{B}(C)$ assigns invariants restricting the admissible time elapse in each location.

The semantics of a TA is given as a TTS as of Definition 1. A state of the corresponding TTS is a pair $(\ell, v) \in L \times \text{Time}^C$, where ℓ is the current location and v a clock valuation. Discrete transitions correspond to edges in E with clock resets, while delay transitions correspond to uniform advancement of all clocks under the invariant of the current location.

Timed automata induce infinite-state timed transition systems due to the continuous evolution of clocks. To enable algorithmic reasoning about these systems, we employ symbolic abstractions based on zones, which provide a finite representation of the underlying timed semantics [33], [34]. A *zone* is a convex set of clock valuations satisfying the same clock constraints. Given a timed automaton (T), its symbolic abstraction is a *zone-based symbolic automaton* $\Gamma(T) = (Q, \rightarrow)$, whose symbolic states in Q are pairs (ℓ, Z) , where $(\ell \in L)$ is a location and (Z) is a zone representing the set of valuations reachable at (ℓ) . Every concrete state (ℓ, v) belongs to a symbolic state (ℓ, Z) such that $(v \in Z)$. Transitions in $(\rightarrow)$ are induced by both discrete actions and time elapse within locations. This construction yields a finite symbolic abstraction that preserves the timed behavior of the original system [33], [34].

VI. RELAXED WEAK TIMED BISIMULATION

The foundation of our refinement definition is built upon timed bisimulation [35], [5], a well-established method for reasoning about behavioral equivalence in real-time systems. Since our primary focus is on externally observable behavior, i.e. any event in $\Sigma_{out} \cup \Sigma_{in}$, rather than internal execution details, we base our refinement approach on weak timed bisimulation (WTB) [5], which abstracts away internal transitions.

We denote by τ any event in $\Sigma_E \setminus (\Sigma_{out} \cup \Sigma_{in})$, and write $s \xrightarrow{\tau} s'$ if $s \xrightarrow{a} s'$ for some such a , with τ^* its reflexive-transitive closure. By construction, $\tau \in \Sigma_E$. WTB is based on the weak transition relation (WTR) of [35], which allows (i) τ -transitions before and after observable events; (ii) τ -transitions to be time-preserving; and (iii) time delays to decompose into sub-delays summing to the original. Formally:

Definition 3. Let $\mathcal{T} = (S, s_0, \Sigma_E, \Sigma_D, \rightarrow)$ be a TTS. Furthermore, let $s, s', s'', s''' \in S$, $\mu \in \Sigma_{out} \cup \Sigma_{in}$, and $\delta, \delta_1, \delta_2 \in \Sigma_D$. The *Weak Transition Relation (WTR)* $\Rightarrow$ is the least relation satisfying:

- 1) $s \xrightarrow{\mu} s'$ iff $\exists s'', s''' \text{ s.t. } s \xrightarrow{\tau^*} s'' \xrightarrow{\mu} s''' \xrightarrow{\tau^*} s'$
- 2) $s \xrightarrow{\delta} s'$ iff $\exists s'', s''' \text{ s.t. } s \xrightarrow{\tau^*} s'' \xrightarrow{\delta} s''' \xrightarrow{\tau^*} s'$
- 3) $s \xRightarrow{\delta} s'$ iff $\exists s'' \text{ s.t. } s \xrightarrow{\delta_1} s'' \xRightarrow{\delta_2} s' \wedge \delta = \delta_1 + \delta_2$.
- 4) Time monotonicity is preserved: If $s \xRightarrow{\delta} s'$, then all intermediate transitions respect $\delta_1, \delta_2 \geq 0$.

We abbreviate $s \xRightarrow{\delta} s'' \xrightarrow{\mu} s'$ as $s \xRightarrow{\delta, \mu} s'$. Although WTR abstracts away internal transitions, WTB still enforces strict timing correspondence between observable events in both systems. In practical system evolution, however, updates frequently introduce buffering or asynchronous input processing, as in Section II, causing inputs to be consumed later than in the original. Such updates preserve all externally observable guarantees, yet WTB rejects them by imposing the same strict timing requirement on received events as on sent events. Since only the latter define contractual obligations to the environment, this symmetry is unnecessarily restrictive. We address this by introducing Relaxed Weak Timed Bisimulation (RWTBS): a refinement relation built on WTR that strictly preserves output timing while permitting controlled relaxation of input reception timing. In the following definition, the expression $(\delta|\mu)^*$ denotes an arbitrary finite sequence of delay and observable event steps. Formally, such a sequence corresponds to a timed path π in the TTS, and the accumulated delay $\sum_{\mu_O}^{\pi} \delta$ is as defined in Section V. Conditions 3 and 4 are understood as quantifying over witnessing timed paths of the respective weak transitions.

Definition 4. Let $\mathcal{T}_A = (S_A, s_{0,A}, \Sigma_{E,A}, \Sigma_{D,A}, \rightarrow_A)$ and $\mathcal{T}_R = (S_R, s_{0,R}, \Sigma_{E,R}, \Sigma_{D,R}, \rightarrow_R)$ be TTSs sharing a common observable interface, i.e. $\Sigma_{out} \cup \Sigma_{in} \subseteq \Sigma_{E,A} \cap \Sigma_{E,R}$. The relation $R \subseteq S_R \times S_A$ is a Relaxed Weak Timed Bisimulation (RWTBS) iff

- 1) Initial State Consistency: $(s_{0,R}, s_{0,A}) \in R$
- 2) Observable Event preservation :

$$\forall \mu \in \Sigma_{out} \cup \Sigma_{in}, \forall (s_R, s_A) \in R :$$

$$s_R \xrightarrow{\delta_R} s'_R \xRightarrow{\mu} s'_A \implies \exists s'_A, \delta_A : s_A \xrightarrow{\delta_A} s'_A \wedge (s'_R, s'_A) \in R$$

$$s_A \xrightarrow{\delta_A} s'_A \xRightarrow{\mu} s'_R \implies \exists s'_R, \delta_R : s_R \xrightarrow{\delta_R} s'_R \wedge (s'_R, s'_A) \in R$$

- 3) Output Timing is Preserved: $\forall \mu_O \in \Sigma_{out} :$

$$\begin{aligned} & \forall (s_R, s_A) \in R, \quad s_R \left(\xrightarrow{\delta'} \Big| \xrightarrow{\mu_R} \right)^* \xRightarrow{\mu_O} s'_R \\ & \implies \exists s'_A : s_A \left(\xrightarrow{\delta} \Big| \xrightarrow{\mu_A} \right)^* \xRightarrow{\mu_O} s'_A \wedge (s'_R, s'_A) \in R \end{aligned}$$

$$\text{such that } \sum_{\mu_O}^{\pi_R} \delta' \leq \sum_{\mu_O}^{\pi_A} \delta$$

4) Relaxed Event Reception Timing: $\forall \mu_I \in \Sigma_{in}$,

$$\begin{aligned} & \forall (s_R, s_A) \in R : s_A \left(\xRightarrow{\delta} \middle| \xRightarrow{\mu_A} \right)^* \xRightarrow{\mu_I} s'_A \\ \implies & \exists s'_R : s_R \left(\xRightarrow{\delta'} \middle| \xRightarrow{\mu_R} \right)^* \xRightarrow{\mu_I} s'_R \wedge (s'_R, s'_A) \in R \\ & \text{such that } \sum_{\mu_I}^{\pi_R} \delta' \geq \sum_{\mu_I}^{\pi_A} \delta, \end{aligned}$$

We write $T_R \overset{rwbs}{\sim} T_A$ when the greatest such relation exists and contains the initial pair (s_{0_R}, s_{0_A}) . With slight abuse of notation, we apply $\overset{rwbs}{\sim}$ both to TTSs and to the TA that induce them: for two TAs we write $T_R \overset{rwbs}{\sim} T_A$ iff the relation holds between their induced TTSs. The level of abstraction is determined by context.

Condition 1 ensures that the initial states of both TTSs are included in the relation, guaranteeing a well-defined starting equivalence. Condition 2 ensures that if a transition exists in one system, a corresponding transition must exist in the other for sent and received events. Condition 3 mandates that the delay for sending an event in the refined system must match that of the abstract system, ensuring that the refined system cannot send events later than specified in the reference model. This restriction preserves the timing guarantees imposed on externally observable outputs. Condition 4 permits the refined system to defer input reception relative to the abstract system, allowing for increased flexibility in event reception timing.

Together, they establish a crucial refinement property: while the refined system is permitted to relax the timing constraints on received events, it must strictly adhere to the original timing guarantees for sent events. This asymmetric relaxation reflects the practical reality of system evolution in real-time systems: while updates may introduce buffering or asynchronous processing that delays input reception, the timing of sent events defines the system's contractual guarantees to the environment and must therefore remain strictly preserved across updates.

While we assume FIFO buffering of received events, the theory is not limited to this specific queuing mechanism. If a different queuing mechanism is required, an auxiliary component model can be introduced to accurately represent the buffer behavior within the refinement verification process.

The alternating send/receive assumption aligns naturally with hard real-time systems, as many real-world communication protocols implement watchdog mechanisms, where transmitted messages must be acknowledged by the receiver within a bounded timeframe [32]. Nevertheless, those are not fundamental requirements of the theory and can be relaxed, as we discuss in Section IX-C.

VII. DECIDING RWTBS

Using TTS as a semantic foundation enables the application of formal verification techniques to reason about refinement. However, the continuous nature of time leads to infinite state spaces, rendering explicit state-space exploration infeasible. Since components are modeled as TA (Definition 2), we exploit their zone-based symbolic abstraction (cf. Section V), yielding finite representations of the component, to decide RWTBS refinement. Let T_R and T_A denote the automata representing the refined and reference components, respectively, and

Algorithm 1: RWTBS Refinement Check

Input: $\Gamma(T_R)$ and $\Gamma(T_A)$
Output: True iff $T_R \overset{rwbs}{\sim} T_A$

```

1 Function Match( $q_R, q_A, \mathcal{R}$ ):
2   forall  $a \in \Sigma_{out} \cup \Sigma_{in}, q'_R \in W_{q_R}^a$  do
3      $W_{q_R}^a \leftarrow \text{cl}_\tau(\text{post}_a(\text{cl}_\tau(q_R)))$ ;
4      $W_{q_A}^a \leftarrow \text{cl}_\tau(\text{post}_a(\text{cl}_\tau(q_A)))$ ;
5     if  $a \in \Sigma_{out}$  then
6       if  $\nexists q'_A \in W_{q_A}^a : \text{Con3}(q_R, q'_R, q_A, q'_A)$  then
7         return false
8     else if  $a \in \Sigma_{in}$  then
9       if  $\nexists q'_A \in W_{q_A}^a : (q'_R, q'_A) \in \mathcal{R}$  then
10        return false
11   forall  $a \in \Sigma_{out} \cup \Sigma_{in}, q'_A \in W_{q_A}^a$  do
12     if  $a \in \Sigma_{in}$  then
13       if  $\nexists q'_R \in W_{q_R}^a : \text{Con4}(a, \delta_R^{\max}, \delta_A^{\max})$  then
14         return false
15     else if  $a \in \Sigma_{out}$  then
16       if  $\nexists q'_R \in W_{q_R}^a : (q'_R, q'_A) \in \mathcal{R}$  then
17         return false
18   return true;
19  $\mathcal{R} \leftarrow Q_R \times Q_A$ ;
20 repeat
21    $\mathcal{R} \leftarrow \{(q_R, q_A) \in \mathcal{R} \mid \text{Match}(q_R, q_A, \mathcal{R})\}$ ;
22 until  $\mathcal{R}$  stabilises;
23 return True iff  $(q_{R,0}, q_{A,0}) \in \mathcal{R}$ ;

```

let Q_R and Q_A be their corresponding sets of symbolic states. The refinement-checking procedure is given in Algorithm 1.

The algorithm maintains a relation $\mathcal{R} \subseteq Q_R \times Q_A$ and iteratively removes state pairs that violate the RWTBS matching conditions until a fixed point is reached. In each iteration, for every pair $(q_R, q_A) \in \mathcal{R}$, the algorithm checks that observable transitions from q_R can be matched by transitions from q_A , and vice versa, in accordance with Definition 4. Matching is implemented by the function $\text{Match}(q_R, q_A, \mathcal{R})$, which computes weak successors for observable actions. Given a symbolic state $q = (\ell, Z)$, weak successors are computed by first constructing the τ -closure $\text{cl}_\tau(q)$, containing all symbolic states reachable via internal transitions and time elapse without producing an observable action. For each state in this closure, the corresponding observable transition is explored, followed by a second τ -closure. The union of the resulting states forms the set of weak successors $\text{WS}(q, a)$. In our implementation, closures and weak-successor sets are memoised and canonical zone states are reused, which avoids recomputing symbolic reachability across validation steps.

Match performs two loops corresponding to the bidirectional matching required by Definition 4.

The first loop iterates over every weak observable step $q_R \xRightarrow{a} q'_R$ in $\Gamma(T_R)$ and requires the existence of a matching step $q_A \xRightarrow{a} q'_A$ in $\Gamma(T_A)$ leading to a successor pair $(q'_R, q'_A) \in \mathcal{R}$. For output events ($a \in \Sigma_{out}$), the matching

step must additionally satisfy the Condition 3 timing constraint (lines 2 — 10); for input events, the corresponding Condition 4 constraint is discharged by the loop in lines 11 — 17.

If any iteration of either loop fails to find a match, Match returns false and the pair (q_R, q_A) is removed from $\mathcal{R}$. Otherwise it returns true and the pair is retained. The main loop iterates until $\mathcal{R}$ stabilises and the algorithm returns true if the initial pair $(q_{R,0}, q_{A,0})$ is related, indicating that $T_R \stackrel{rw\text{tbs}}{\sim} T_A$.

Theorem 1. *The worst-case time complexity of Algorithm 1 is $O(P^2 + R(|E|_R) + R(|E|_A))$, where P is the number of candidate pairs and $R(|E|)$ is the cost of computing weak successors for $|E|$ transitions.*

Proof. In the appendix. $\square$

Theorem 2. *Algorithm 1 terminates and returns true if and only if $T_R \stackrel{rw\text{tbs}}{\sim} T_A$.*

Proof. In the appendix. $\square$

When considering the practical performance of the algorithm, it is important to note that the pair validation is naturally parallelisable. Symbolic explorations for distinct states and action labels are independent, as are pair checks. With k available cores, the effective cost can be reduced by up to a factor of k in practice, subject to memory bandwidth limits.

Compared to full re-verification, which requires explicit exploration of all reachable states, leading to exponential growth with system size [36], refinement checking is significantly more efficient.

Example 1. We illustrate how RWTBS enables the update from Section II, which fails under WTBS. We model the originally deployed system as $T_A = (L_A, \ell_{0,A}, \{x\}, \Sigma_E, E_A, \text{Inv}_A)$ with $L_A = \{i_A, p_A, d_A\}$, $\ell_{0,A} = i_A$, $E_A = \{i_A \xrightarrow{\text{gps_fix?}, x:=0} p_A, p_A \xrightarrow{\text{waypoint!}} d_A, d_A \xrightarrow{\tau, x:=0} i_A\}$, $\text{Inv}_A(i_A) = (x \leq 0)$, and $\text{Inv}_A(p_A) = (x \leq 100)$. The invariant on i_A encodes the physical validity window of a GPS fix, which is valid only at the instant of acquisition. The induced TTS $\mathcal{T}_A$ satisfies $\sum_{\text{gps_fix?}} \delta_A = 0$ and $\sum_{\text{waypoint!}} \delta_A \leq 100$. The updated system is modeled as $T_R = (L_R, \ell_{0,R}, \{x\}, \Sigma_E, E_R, \text{Inv}_R)$ with $L_R = \{i_R, p_R, d_R\}$, $\ell_{0,R} = i_R$, $E_R = \{i_R \xrightarrow{\text{gps_fix?}, x:=0} p_R, p_R \xrightarrow{\text{waypoint!}} d_R, d_R \xrightarrow{\tau, x:=0} i_R\}$, $\text{Inv}_R(i_R) = (x \leq 50)$, and $\text{Inv}_R(p_R) = (x \leq 50)$. The relaxed invariant on i_R reflects the 50 ms validity window enabled by fusion; the tightened invariant on p_R reflects the additional fusion computation. The induced TTS $\mathcal{T}_R$ satisfies $\sum_{\text{gps_fix?}} \delta_R \leq 50$ and $\sum_{\text{waypoint!}} \delta_R \leq 50 + 50 = 100$. WTBS requires equal accumulated delays for all matched observable transitions. Since $\mathcal{T}_R$ admits firing gps_fix? at any $x \in [0, 50]$ while $\text{Inv}_A(i_A) = (x \leq 0)$ admits only $x = 0$, and τ -transitions are time-preserving and cannot introduce new time elapse, no weak transition in $\mathcal{T}_A$ from i_A matches the deferred consumption in $\mathcal{T}_R$. Hence $T_R \not\stackrel{wtb}{\sim} T_A$.

Applying Algorithm 1 to T_A and T_R yields a symbolic witness relation over their zone graphs, which, projected to their respective locations, results in $\stackrel{rw\text{tbs}}{\sim}$

$\{(i_R, i_A), (p_R, p_A), (d_R, d_A)\}$. We verify that $\stackrel{rw\text{tbs}}{\sim}$ is a RWTBS relation. Condition 1 holds by construction since $(i_R, i_A) \in \stackrel{rw\text{tbs}}{\sim}$. For Condition 2, $i_R \xrightarrow{\text{gps_fix?}} p_R$ is matched by $i_A \xrightarrow{\text{gps_fix?}} p_A$ with $(p_R, p_A) \in \stackrel{rw\text{tbs}}{\sim}$, and $p_R \xrightarrow{\text{waypoint!}} d_R$ is matched by $p_A \xrightarrow{\text{waypoint!}} d_A$ with $(d_R, d_A) \in \stackrel{rw\text{tbs}}{\sim}$; backward directions are symmetric. Condition 3 holds since for every path in $\mathcal{T}_R$ with $\sum_{\text{waypoint!}} \delta_R \in [0, 100]$ there is a matching path in $\mathcal{T}_A$ with $\sum_{\text{waypoint!}} \delta_A = \sum_{\text{waypoint!}} \delta_R$, preserving output timing. Condition 4 holds since $\sum_{\text{gps_fix?}} \delta_R \leq 50 \geq 0 = \sum_{\text{gps_fix?}} \delta_A$, satisfying relaxed input reception. By Theorem 2, this witnesses $T_R \stackrel{rw\text{tbs}}{\sim} T_A$.

VIII. PRESERVATION OF PROPERTIES

In the following, we present key preservation results for RWTBS. We first observe that Condition 2 of Definition 4 directly enforces matching weak transitions for every $\mu \in \Sigma_{\text{out}} \cup \Sigma_{\text{in}}$, independently of timing. Consequently, $\stackrel{rw\text{tbs}}{\sim}$ witnesses a weak bisimulation over observable events, hence implies weak trace equivalence over $\Sigma_{\text{out}} \cup \Sigma_{\text{in}}$, and all next-free linear-time properties over observable events are preserved if $\mathcal{T}_R \stackrel{rw\text{tbs}}{\sim} \mathcal{T}_A$.

In contract-based design, timing requirements on output events constitute guarantees to the environment. Specifically, given satisfied assumptions, a component must emit outputs within specified deadlines. Under RWTBS, this is enforced by Condition 3. We first establish that this is sufficient to preserve output deadlines in refined implementations.

Lemma 1. *Let $(s_R, s_A) \in \stackrel{rw\text{tbs}}{\sim}$. If for every path in $\mathcal{T}_A$ from s_A , every output event $\mu_O \in \Sigma_{\text{out}}$ occurs within deadline $D \in \text{Time}$, then for every path in $\mathcal{T}_R$ from s_R , every μ_O also occurs within deadline D .*

Proof. In the appendix. $\square$

Using Lemma 1, we can now establish that under RWTBS a refined component can safely replace its predecessor without violating the original contract.

Theorem 3. *Let $C = (A, G)$ be a contract. Let $\mathcal{T}_A$ and $\mathcal{T}_R$ be the TTSs with $\mathcal{T}_R \stackrel{rw\text{tbs}}{\sim} \mathcal{T}_A$. If $\mathcal{T}_A \models C$, then $\mathcal{T}_R \models C$.*

Proof. In the appendix. $\square$

Corollary 1 extends Theorem 3 by characterising the maximum input delay configurable in a candidate refinement while still guaranteeing contract preservation. To this end, we introduce the following notation. For any $\mu_O \in \Sigma_{\text{out}}$ and any state s of a TTS, we denote with $D(s, \mu_O)$ the maximum accumulated delay over all paths from s to μ_O :

$$D(s, \mu_O) = \max \left\{ \sum \delta \mid s \left(\xrightarrow{\delta} \right)^* \xrightarrow{\mu_O} s' \right\}$$

where $\mu \in \Sigma_{\text{out}} \cup \Sigma_{\text{in}}$.

Corollary 1. *Let $\mathcal{T}_R \stackrel{rw\text{tbs}}{\sim} \mathcal{T}_A$ and C be a contract such that $\mathcal{T}_A \models C$, where each $\mu_O \in \Sigma_{\text{out}}$ is constrained by a deadline D_{μ_O} measured from a designated reference event $\mu \in \Sigma_E$. For any post-reception pair $(s_R, s_A) \in \stackrel{rw\text{tbs}}{\sim}$, let t_{elapsed} denote the time elapsed since μ upon reaching s_R . Then the*

maximum input delay preserving $\mathcal{T}_R \models \mathcal{C}$ is $\delta_{\max}(s_R) = \min_{\mu_O \in \Sigma_{out}(s_R, s_A)} [D_{\mu_O} - t_{\text{elapsed}} - D(s_R, \mu_O)]$, where $\Sigma_{out}(s_R, s_A) \subseteq \Sigma_{out}$ denotes the output events reachable from (s_R, s_A) .

Proof. In the appendix. $\square$

Engineers can use this insight to perform design space exploration analytically, directly at the view level using nominal parameters derived from component datasheets.

Example 2. Consider Example 1 and the contract imposing a 100 ms deadline on *waypoint!* from GPS-fix acquisition. For the post-consumption pair $(p_R, p_A) \in \sim^{rwts}$, we have $t_{\text{elapsed}} = 0$ ms since reaching p_R requires only the zero-delay consumption transition. From $Inv_R(p_R) = (x \leq 50)$ and the clock reset on entry to p_R , the worst-case remaining delay is $D(p_R, \text{waypoint!}) = 50$ ms. Corollary 1 therefore yields $\delta_{\max} = 100 - 0 - 50 = 50$ ms. Hence, any deferred-consumption configuration with $\delta_I \leq 50$ ms preserves the waypoint deadline by Theorem 3, without requiring re-verification.

One of the most desirable properties in system verification is compositionality: the ability to verify components independently while ensuring that system-level guarantees are preserved under composition. In our framework, component interactions are mediated by contracts (Section V), and system-level guarantees are accordingly expressed as composed contracts. Compositionality of RWTBS is therefore most naturally established at the contract level: if each refined component preserves its component-level contract, the composed refined system preserves the composed contract.

Theorem 4. Let M_1 and M_2 be two components and let $\mathcal{T}_{A1}, \mathcal{T}_{A2}$ and $\mathcal{T}_{R1}, \mathcal{T}_{R2}$ be their respective abstract and refined TTSs, such that $\mathcal{T}_{R1} \sim^{rwts} \mathcal{T}_{A1}$ and $\mathcal{T}_{R2} \sim^{rwts} \mathcal{T}_{A2}$. Let $C_1 = (A_1, G_1)$ and $C_2 = (A_2, G_2)$ be compatible contracts such that $\mathcal{T}_{A1} \models C_1$ and $\mathcal{T}_{A2} \models C_2$. Then $\mathcal{T}_{R1} \parallel \mathcal{T}_{R2} \models C_1 \otimes C_2$.

In some applications common in real-time systems, such as time-triggered communication [37] and hardware-timed actuation [32], it is desirable to preserve the exact timing of sent events rather than just upper bounds. In this special case, Condition 3 can be strengthened from $\delta_R \leq \delta_A$ to $\delta_R = \delta_A$, yielding a timed weak bisimulation on sent events and implying preservation of all TCTL properties over the output interface [5].

Theorem 5. Let $\mathcal{T}_A$ and $\mathcal{T}_R$ be TTSs with $\mathcal{T}_R \sim^{rwts} \mathcal{T}_A$. Assume that for every $(s_R, s_A) \in \sim^{rwts}$ and every $\mu_O \in \Sigma_{out}$, the matching of Condition 3 holds in both directions with $\sum_{\mu_O} \delta = \sum_{\mu_O} \delta'$. Then $\sim^{rwts}$ is a WTB on output events.

Proof. In the appendix. $\square$

IX. EVALUATION

This section evaluates our refinement approach through two complementary studies grounded in a smart farming initiative developed within a large national research programme on autonomous agriculture and environmental sustainability. To

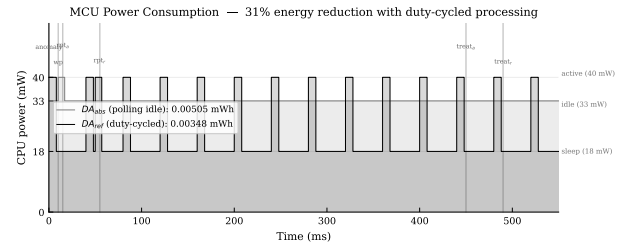
this end, we present a case study to demonstrate its practical applicability and engineering relevance (Section IX-A) and an experiment to compare its performance against the state-of-the-art alternative of model-checking (Section IX-B).

A. Sugar Beet Field Monitoring Fleet Case Study

The purpose of this case study is to demonstrate the applicability of RWTBS in a representative real-time scenario, showing that it accepts a valid update which would be rejected by classical timed refinement, and to illustrate its engineering utility: how the formal guarantees of RWTBS can be used to characterise the safe operating envelope of an update, providing engineers with quantifiable insights into the trade-offs between different key system parameters such as energy savings and timing guarantees directly at the view level, without hardware deployment.

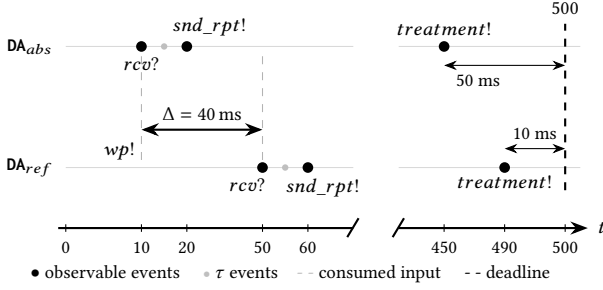
1) *Scenario Description:* A fleet of autonomous systems is deployed for sugar beet field monitoring. The fleet comprises three components: a Fleet Coordinator (FC) that plans missions, dispatches waypoints, and issues treatment commands; a Drone Agent (DA) that performs aerial anomaly detection that runs on a STM32L4-class MCU [38] and transmits reports to FC; and a Spraying Drone (SD) that applies targeted pesticide treatment upon receiving commands from FC. The FC, DA and SD contracts are compatible by construction, since each component's output guarantees discharge the input assumptions of its downstream consumer. The end-to-end contractual deadline from anomaly detection to issuing of the treatment command is 500 ms.

Figure 2: CPU power consumption over time.



2) *Verification of a Component Update:* We model a first implementation of the DA, denoted DA_{abs} , using an interrupt-driven architecture for receiving waypoint messages from FC. As shown in Figure 2, this architecture keeps CPU active and the radio receiver armed at all times, triggering immediate processing upon arrival, depicted as an energy spike corresponding to the *snd_rpt_a* event. This model refines the system view, i.e. is verified against the DA's functional and timing behavior, including the 500 ms deadline. Following deployment, excessive energy consumption was identified as an operational bottleneck. Engineers, therefore, derived a virtual view based on nominal STM32L4-class MCU parameters [38], in the form of a Python simulation to develop an energy-saving update. As shown in Figure 2, DA_{abs} keeps the CPU in Run mode at 80 MHz with the radio receiver always armed, even during idle periods when no waypoints are being sent by FC. The virtual view shows that at 3.3 V, DA_{abs} draws approximately 33 mW

Figure 3: Witness trace for classical bisimulation rejection.



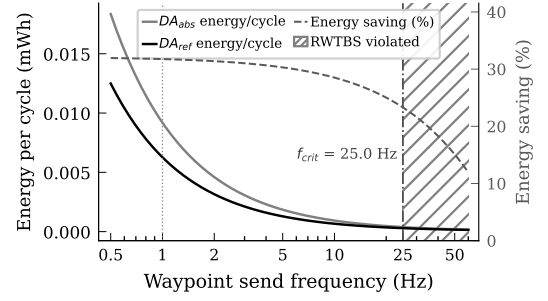
continuously (~ 8 mA CPU + ~ 2 mA radio RX). Accordingly, a duty-cycled processing update DA_{ref} was developed using the virtual view. As shown in Figure 2, DA_{ref} duty-cycles the CPU with a 40 ms period and an 8 ms wake window: during sleep, the MCU sleeps with the RTC running and the radio transceiver in quiescent standby, drawing approximately 18 mW (~ 5.5 mA total: CPU ~ 1.4 μ A + radio standby ~ 3 mA + TCXO ~ 2.5 mA). The event snd_rpt_r is now preceded by a duty-cycle wake event, yielding a 31% reduction in energy consumption over a representative mission window. By construction, DA_{ref} is a valid refinement of the virtual views.

3) *RWTBS vs. WTB*: As discussed in Section IV, safe deployment requires establishing that DA_{ref} refines DA_{abs} . In this scenario, WTB fails to establish this relationship, rejecting the update as unsafe. To see why, consider the execution fragment shown in Figure 3. The FC dispatches a waypoint at $t = 10$ ms. In DA_{abs} , the message is consumed immediately upon arrival — the input action wp fires at $t = 10$ ms. In DA_{ref} , the CPU is sleeping and only wakes at the next duty-cycle boundary, consuming the message at $t = 50$ ms. WTB requires exact timing correspondence for all observable events, including inputs: since wp fires at $t = 10$ ms in DA_{abs} but at $t = 50$ ms in DA_{ref} , the update is therefore rejected by WTB. Yet both systems issue $snd_rpt!$ before the 500 ms end-to-end deadline, respecting the contractual obligation in both cases.

RWTBS resolves this by treating input and output timing asymmetrically. The 40 ms input consumption delay is permitted by Condition 4. The 500 ms output deadline is strictly preserved by Condition 3. Contract satisfaction transfers to the full composed system $FC \parallel DA \parallel SD$ by Theorem 4: each component's RWTBS-verified contract is preserved under refinement (Theorem 3), and composition of the component contracts yields the system-level guarantee without re-verifying the parallel composition.

4) *Safe Operating Envelope*: Corollary 1 provides an explicit characterisation of the maximum duty-cycle delay configurable while still guaranteeing contract preservation. Letting μ_O be the treatment command, it directly yields: $\delta_{max} = D_{\mu_O} - t_{elapsed} - D(s_R, \mu_O) = 500 - 10 - 440 = 50$ ms, where $D_{\mu_O} = 500$ ms is the contract deadline on the treatment command (measured from anomaly detection), $t_{elapsed} = 10$ ms is the time already elapsed when DA reaches the post-reception state s_R (the FC waypoint-dispatch latency), and $D(s_R, \mu_O) = 440$ ms is the maximum accumulated delay along any path from s_R to μ_O , comprising 5 ms of DA

Figure 4: Energy per cycle (left axis) and energy saving percentage (right axis) function of waypoint send frequency



processing and 435 ms of FC pipeline execution. Any configured duty-cycle delay $\delta_I \leq \delta_{max}$ is guaranteed to preserve contract satisfaction by Theorems 3 and 4, without requiring re-verification. In fact, DA_{ref} with $\delta_{ref} = 40$ ms lies 10 ms inside δ_{max} and is correctly accepted by RWTBS.

In our case study, Corollary 1 provides a closed-form bound on the maximum configurable duty-cycle delay that guarantees contract preservation, enabling engineers to select an optimal $\delta_I \leq \delta_{max}$ without hardware deployment or re-verification.

Figure 4 shows that the duty-cycled architecture achieves greater per-cycle energy savings at lower frequencies, where more time is spent in the 18 mW sleep state instead of the 33 mW idle baseline. At the 1 Hz case-study operating point, the refined system reduces per-cycle energy consumption by $\sim 31\%$, and RWTBS certifies that this optimisation remains safe across the full operating envelope $f \in (0, 25]$ Hz. The saving degrades from 31.9% at 0.5 Hz to 23.3% at 25 Hz, where the duty-cycle period equals the inter-message interval. Beyond this point, Condition 4 is violated and the architecture is formally unsafe by Theorems 3 and 4. Without RWTBS, an engineer would have no formal characterisation of where in this envelope the optimisation remains safe; the only alternatives are full re-verification at each operating frequency, or empirical testing on hardware.

This demonstrates that RWTBS provides engineers with strictly more insight than classical timed bisimulation. Qualitatively, it accepts the update that classical bisimulation rejects. Quantitatively, it characterises the full safe configuration space $\delta_I \in [0, 50]$ ms, enabling informed engineering trade-offs rather than a binary verdict. This decouples design space exploration from hardware availability, reduces iteration cost, and provides a formal guarantee that the chosen configuration is safe, something neither classical refinement nor model checking alone can offer at this granularity.

B. Performance Evaluation: RWTBS vs. Full Model Checking

This subsection compares the performance of RWTBS refinement checking against full model checking. Compositional refinement checkers such as ECDAR [18] enforce symmetric delay matching: any update that defers input reception relative to the abstract system is rejected regardless of whether output deadlines are preserved (cf. Section III). ECDAR would therefore return a negative verdict on every configuration in our

Table I: Summary of evaluation benchmarks.

Configuration	Locations	μ_O	μ_I	τ -trans.	Clocks	Zones	$c_{\max}$
Small	12	3	3	28	2	478	28
Small (3 Comp)	36	9	7	86	6	997	28
Small (5 Comp)	60	15	11	144	10	1605	28
Small (7 Comp)	84	21	15	202	14	2132	28
Medium	14	3	3	34	2	743	30
Medium (3 Comp)	42	15	13	92	6	1427	30
Medium (5 Comp)	70	25	21	154	10	2089	30
Medium (7 Comp)	98	35	29	216	14	2730	30
Large	17	3	3	43	2	964	29
Large (3 Comp)	51	21	19	107	6	2322	29
Large (5 Comp)	85	35	31	179	10	3540	29
Large (7 Comp)	119	49	43	251	14	4223	29
Extra Large	19	3	3	49	2	1467	28
Extra Large (3 Comp)	57	27	25	113	6	2952	28
Extra Large (5 Comp)	95	45	41	189	10	5008	28
Extra Large (7 Comp)	133	63	57	265	14	6502	28
Extra Extra Large	22	3	3	58	2	1875	26
Extra Extra Large (3 Comp)	66	33	31	128	6	4506	26
Extra Extra Large (5 Comp)	110	55	51	214	10	7049	26
Extra Extra Large (7 Comp)	154	77	71	300	14	9460	29

evaluation without rendering a performance comparison uninformative. Full model checking, by contrast, can be used to verify that the update satisfies the desired properties, making it the only existing workflow able to validate the class of updates that RWTBS addresses. The performance gap reported below therefore reflects the cost difference between compositional refinement checking and this alternative.

1) *Methodology*: The evaluation spans 20 system configurations grouped into five size categories, each tested under single-component, 3-, 5-, and 7-component parallel compositions, as summarized in Table I. Each configuration was generated using a parameterized model generator with a fixed random seed for reproducibility. The generator constructs strongly connected timed automata with a backbone cycle and additional branching edges, using two clocks per component (one for invariants, one for guards), deterministic one-to-one channel synchronization, and simple clock bounds without clock-difference constraints. The configurations were designed to be representative of realistic smart farming models used in the project. All experiments were performed on an Ubuntu-based virtual machine under KVM with 8 vCPUs and 128GB of RAM. Algorithm 1 has been implemented in C++, with parallelism achieved via OpenMP. The approach is compared against UPPAAL [39] as the representative model checker, which requires cross-product composition of all components for system-wide verification. The approach is representative of any explicit-state method requiring full state-space exploration, including PRISM [40] and KRONOS [41]. We report verification time and memory consumption across all configurations, with a ten-hour time limit. RWTBS checks are performed on all components in isolation and averaged across 3 runs.

2) *Results*: Table II reports our results. For single-component systems, RWTBS and Uppaal are broadly comparable, with Uppaal holding a modest advantage on the Extra Large and Extra Extra Large configurations over the parallel implementation, reflecting the expected overhead of symbolic matching relative to direct state-space exploration when no compositional savings are available. The break-even point lies at 3 components, beyond which compositional verification eliminates the need to construct a global product automaton and the cost structure reverses decisively. The Extra Extra Large (3 Comp) configuration is a notable exception: Uppaal

Table II: Comparison of Verification Time and Memory Footprint. Best values are highlighted in bold.

Configuration	Time (s)			Memory (KB)	
	RWTBS	RWTBS (Paral.)	UPPAAL	RWTBS	UPPAAL
Small	0.063	0.03	0.101	6.23	18024
Medium	0.16	0.059	0.101	9.13	17920
Large	0.24	0.074	0.101	11	17824
Extra Large	0.316	0.106	0.101	16.1	17992
Extra Extra Large	0.685	0.177	0.101	22.9	18188
Small (3 Comp)	0.1	0.047	0.101	12.8	18608
Medium (3 Comp)	0.16	0.069	1.003	18.1	30456
Large (3 Comp)	0.277	0.127	0.402	27.3	23016
Extra Large (3 Comp)	0.291	0.193	0.502	33.4	24636
Extra Extra Large (3 Comp)	0.68	0.361	0.201	55.2	20868
Small (5 Comp)	0.159	0.076	45.619	20.5	223540
Medium (5 Comp)	0.203	0.105	> 36000*	26.1	> 81216*
Large (5 Comp)	0.42	0.195	> 36000*	42.7	> 81216*
Extra Large (5 Comp)	0.558	0.307	> 36000*	58.2	> 81216*
Extra Extra Large (5 Comp)	1.034	0.527	> 36000*	86.1	> 81216*
Small (7 Comp)	0.194	0.094	> 36000*	26.2	> 81216*
Medium (7 Comp)	0.238	0.128	> 36000*	33.9	> 81216*
Large (7 Comp)	0.464	0.217	> 36000*	51.1	> 81216*
Extra Large (7 Comp)	0.702	0.389	> 36000*	75.3	> 81216*
Extra Extra Large (7 Comp)	1.43	0.776	> 36000*	113	> 81216*

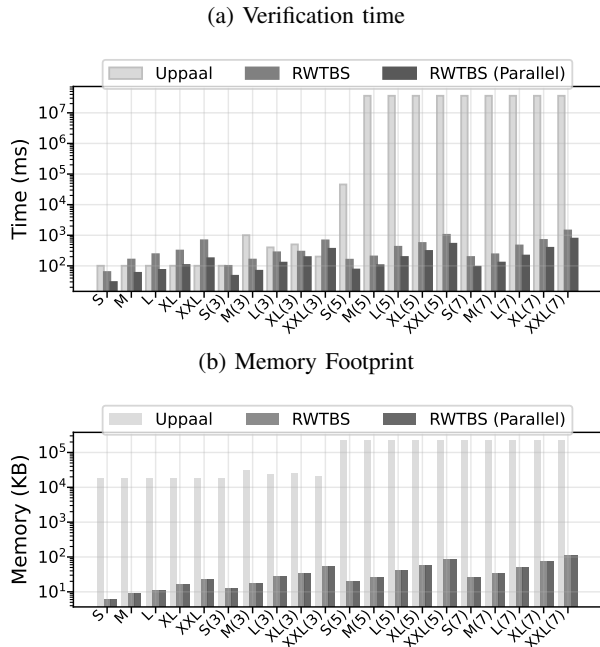
*UPPAAL verification exceeded 10 hours and was terminated.

completes in 0.20 s against 0.36 s for parallel RWTBS, owing to the smallest $c_{\max}$ in the benchmark suite (26), which keeps Uppaal's zone graph compact, combined with a three-component product that remains small enough to avoid cross-product blow-up.

As the number of components grows, the advantages of RWTBS become increasingly pronounced, exhibiting a qualitative phase transition between the three- and five-component configurations. For three-component systems, except the abovementioned Extra Extra Large configuration, the parallel RWTBS implementation outperforms Uppaal by up to $15\times$, with the serial implementation remaining competitive across all but the largest configurations. In five-component systems, Uppaal completes only the Small configuration within the ten-hour time limit, whereas RWTBS completes all five configurations in under ~ 500 ms for the parallel implementation, which represents a reduction of up to five orders of magnitude in verification time. This improvement is not a consequence of implementation differences between tools: the exponential blow-up in Uppaal is an inherent consequence of any approach requiring explicit global state-space construction via cross-product composition, as the EXPTIME-complete complexity of full TCTL model checking predicts [39]. PRISM and KRONOS would exhibit the same asymptotic behaviour. In contrast, RWTBS verifies components independently and reuses cached symbolic successors, yielding polynomial growth in the size of the largest component, which is consistent with Theorem 1 and directly reflecting the compositionality guarantee of Theorem 4. The parallelised implementation achieves consistent additional speedups of approximately $4\text{--}5\times$ over the serial version across all multi-component configurations, with the serial version already dominating Uppaal by several orders of magnitude, confirming that parallelism provides a bonus rather than a prerequisite.

Memory usage tells an equally compelling story. RWTBS peaks at 113 KB for the largest seven-component configuration, against >81 MB for Uppaal in comparable configurations, representing a reduction of roughly three orders of magnitude, as illustrated in Figure 5b. This gap arises from the same structural difference: Uppaal must materialise the

Figure 5: Re-verification vs. RWTBS across different configurations. The y-axis is logarithmic to enhance readability.



global product state space in memory, whereas RWTBS maintains only the zone representations of individual components. The resulting footprint, therefore, remains small even for the largest configurations. This opens the prospect of on-device refinement checking: rather than routing proposed over-the-air updates to offline verification infrastructure, a deployed system could carry its own verification oracle and perform incremental self-validation locally before committing to an update.

The contrast in performance across multi-component configurations underscores both the scalability of RWTBS and its applicability to realistic autonomous systems, which routinely comprise multiple interacting components each with its own timing constraints and behavioral guarantees. The ability to verify adaptations without incurring the prohibitive costs of full model checking is crucial for ensuring timely and reliable updates in such settings. When only a single component requires adaptation, RWTBS verifies the change in complete isolation, without touching the remainder of the state space. This represents a particularly valuable property for large systems, where the cost of full model checking would be prohibitive even for a single component update.

C. Discussion and Threats to Validity

The benchmark configurations were generated synthetically to enable controlled, reproducible scaling across component counts and system sizes. This approach isolates the compositional advantage of RWTBS from confounding factors present in ad-hoc model collections. The generator parameters were calibrated against the structural properties of the case study fleet models. These parameters are conservative with respect to state-space complexity: systems with more clocks or complex guard structures would disproportionately increase the cost

of full model checking relative to compositional refinement checking, likely widening the reported performance gap.

The case study models assume FIFO-buffered input ports with at most one pending message per port at any time, consistent with the modelling choices of Section V and typical of hard real-time systems, as discussed in Section VI. This alternating send/receive constraint is a deliberate modelling choice rather than a fundamental limitation of RWTBS: it bounds buffer occupancy to 1, which is the condition under which Corollary 1 yields a finite, path-independent $\delta_{\max}$. The per-step matching of Definition 4 and the preservation results of Section VIII remain unaffected by dropping it, but without alternation a component may accumulate multiple unconsumed inputs before producing an output, causing reception delays to compound across successive events. In this setting, the bound of Corollary 1 must be re-derived as a function of the entire reception history. Supporting alternative queuing schemes or concurrent multi-message protocols would require extending the system models with additional clocks, synchronization, and auxiliary components representing the new buffering behaviour. We leave the formal treatment of the general multi-message case to future work.

X. SUMMARY AND OUTLOOK

In this paper, we introduced RWTBS, a novel refinement relation for timed systems that bridges the gap between idealized symmetric timing assumptions and the realities of evolving cyber-physical systems. By grounding the relation in WTRs and treating input and output timing asymmetrically, RWTBS accommodates practical adaptations, such as buffering and duty-cycled processing, while strictly preserving the externally observable guarantees that define a system's contractual obligations. We established that RWTBS maintains key properties including compositional contract preservation, and presented a symbolic refinement-checking algorithm that achieves this with polynomial complexity and memory footprints orders of magnitude smaller than full model checking.

To further enhance the applicability and scalability of RWTBS, several research directions remain open. Integrating RWTBS into established engineering environments, such as ROS-based simulation frameworks would enable seamless runtime verification of dynamically evolving system models. Extending RWTBS to probabilistic timed automata or stochastic execution models would allow refinement checking under quantified uncertainty, bridging the gap between worst-case timing guarantees and statistically observed execution behavior. Another promising direction is the combination of RWTBS with runtime monitoring, allowing execution traces to validate refinement assumptions and trigger re-verification when violations occur.

ACKNOWLEDGMENT

Supported by the Ministry of Economic Affairs, Industry, Climate Action and Energy of the State of North Rhine-Westphalia. Co-funded by the European Union. Grant number EFRE-20800498.

REFERENCES

- [1] P. Grimmeisen, A. Wortmann, and A. Morozov, "Case study on automated and continuous reliability assessment of software-defined manufacturing based on digital twins," in *Proceedings of the 25th International Conference on Model Driven Engineering Languages and Systems: Companion Proceedings*, ser. MODELS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 511–518.
- [2] P. Derler and E. Lee, "Modeling cyber-physical systems," *Proceedings of the IEEE*, vol. 100, pp. 13–28, 01 2012.
- [3] J. C. Jensen, D. H. Chang, and E. A. Lee, "A model-based design methodology for cyber-physical systems," in *2011 7th International Wireless Communications and Mobile Computing Conference*, 2011, pp. 1666–1671.
- [4] L. Brodo, G. Scalora, and S. Henkler, "Property refinement in linear temporal logic: Formal semantics and algorithms for software verification," in *Proceedings of the ACM International Conference on the Foundations of Software Engineering*, ser. FSE '26. ACM, July 2026, to appear. [Online]. Available: <https://conf.researchr.org/home/fse-2026>
- [5] S. Tripakis and S. Yovine, "Analysis of timed systems using time-abstraction bisimulations," *Formal Methods in System Design*, vol. 18, no. 1, pp. 25–68, 2001.
- [6] J. Bengtsson and W. Yi, "Timed automata: Semantics, algorithms and tools," in *International School on Formal Methods for the Design of Computer, Communication and Software Systems: Executable Modeling Techniques*, ser. Lecture Notes in Computer Science. Springer, 2004, vol. 3185, pp. 87–124.
- [7] S. Guha, C. Narayan, and S. Arun-Kumar, "On decidability of prebisimulation for timed automata," in *Computer Aided Verification*, P. Madhusudan and S. A. Seshia, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 444–461.
- [8] A. Benveniste, B. Caillaud, D. Nickovic, R. Passerone, J. Raclet, P. Reinkemeier, A. Sangiovanni-Vincentelli, W. Damm, T. A. Henzinger, and K. G. Larsen, "Contracts for systems design: Theory," in *Contracts for Systems Design*, ser. Lecture Notes in Computer Science. Springer, 2014, vol. 8856, pp. 51–92.
- [9] E. M. Clarke, O. Grumberg, and D. A. Peled, *Model Checking*. MIT Press, 2000.
- [10] F. Abbasi, P. Brimont, C. Pruski, and J.-S. Sottet, "Understanding semantic drift in model driven digital twins," in *Proceedings of the ACM/IEEE 27th International Conference on Model Driven Engineering Languages and Systems*, ser. MODELS Companion '24. New York, NY, USA: Association for Computing Machinery, 2024, p. 419–430. [Online]. Available: <https://doi.org/10.1145/3652620.3688256>
- [11] G. Blair, N. Bencomo, and R. B. France, "Models@run.time," *Computer*, vol. 42, no. 10, pp. 22–27, 2009.
- [12] A. Bauer, M. Leucker, and C. Schallhart, "Runtime verification for ltl and tltl," *ACM Computing Surveys*, vol. 43, no. 4, pp. 1–33, 2011.
- [13] C. Steinmetz, G. N. Schroeder, A. Sulak, K. Tuna, A. Binotto, A. Retberg, and C. E. Pereira, "A methodology for creating semantic digital twin models supported by knowledge graphs," in *2022 IEEE 27th International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE. Stuttgart, Germany: IEEE, 2022, pp. 1–7.
- [14] D. Lehner, S. Sint, M. Vierhauser, W. Narzt, and M. Wimmer, "Aml4dt: A model-driven framework for developing and maintaining digital twins with automationml," in *2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA)*, IEEE. Zaragoza, Spain: IEEE, 2021, pp. 1–8.
- [15] E. Kamburjan, C. C. Din, R. Schlatter, S. L. T. Tarifa, and E. B. Johnsen, "Twinning-by-construction: Ensuring correctness for self-adaptive digital twins," in *Leveraging Applications of Formal Methods, Verification and Validation. Verification Principles: 11th International Symposium, ISOFA 2022, Rhodes, Greece, October 22–30, 2022, Proceedings, Part I*. Berlin, Heidelberg: Springer-Verlag, 2022, p. 188–204. [Online]. Available: https://doi.org/10.1007/978-3-031-19849-6_12
- [16] L. de Alfaro and T. A. Henzinger, "Interface automata," in *Proceedings of the 8th European Software Engineering Conference held jointly with 9th ACM SIGSOFT International Symposium on Foundations of Software Engineering (ESEC/FSE)*. ACM, 2001, pp. 109–120.
- [17] A. David, K. G. Larsen, A. Legay, U. Nyman, and A. Wasowski, "Timed i/o automata: a complete specification theory for real-time systems," in *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, ser. HSCC '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 91–100. [Online]. Available: <https://doi.org/10.1145/1755952.1755967>
- [18] A. L. Alexandre David, Kim G. Larsen, U. Nyman, and A. Wasowski, "ECDAR: An environment for compositional design and analysis of real time systems," in *Proceedings of Automated Technology for Verification and Analysis*, ser. LNCS, vol. 6252. Springer, 2010, pp. 365–370.
- [19] L. Brodo, G. Scalora, and S. Henkler, "Property refinement in linear temporal logic: Formal semantics and algorithms for software verification," *Proc. ACM Softw. Eng.*, vol. 3, no. FSE, pp. 1–22, Jun. 2026. [Online]. Available: <https://doi.org/10.1145/3808127>
- [20] —, "What do we really need? identifying minimal sufficient temporal properties via refinement," in *Proceedings of the IEEE/ACM 48th International Conference on Software Engineering: Companion Proceedings (ICSE-Companion '26)*. Rio de Janeiro, Brazil: ACM, 2026, pp. 1–2, to appear in 2026.
- [21] O. Sankur, "Robustness in timed automata: Analysis, synthesis, implementation," Ph.D. dissertation, L'Ecole Normale Supérieure de Cachan, 06 2013.
- [22] K. G. Larsen, A. Legay, L.-M. Traonouez, and A. Wasowski, "Robust synthesis for real-time systems," *Theoretical Computer Science*, vol. 515, pp. 96–122, 2014.
- [23] L. P. Carloni, K. L. McMillan, and A. L. Sangiovanni-Vincentelli, "Latency-insensitive design," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, no. 9, pp. 1059–1076, 2001.
- [24] S. Chakraborty and M. R. Greenstreet, "Elastic circuits," in *Proceedings of the 46th Design Automation Conference (DAC)*, 2009, pp. 66–71.
- [25] T. A. Henzinger, R. Jhala, and R. Majumdar, "Quantitative reactive modeling," in *Proceedings of the 16th International Conference on Concurrency Theory (CONCUR)*, ser. LNCS, vol. 3653. Springer, 2005, pp. 549–563.
- [26] K. Chatterjee and T. A. Henzinger, "Games on timed automata," in *Proceedings of the 13th International Conference on Hybrid Systems: Computation and Control (HSCC)*. ACM, 2010, pp. 101–110.
- [27] K. Pohl, H. Hönniger, R. Achatz, and M. Broy, Eds., *Model-Based Engineering of Embedded Systems: The SPES 2020 Methodology*, 1st ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, eBook ISBN 978-3-642-34614-9; Softcover ISBN 978-3-642-43992-6. [Online]. Available: <https://doi.org/10.1007/978-3-642-34614-9>
- [28] M. Hirsch, S. Henkler, and H. Giese, "Modeling collaborations with dynamic structural adaptation in mechatronic uml," in *Proceedings of the 2008 International Workshop on Software Engineering for Adaptive and Self-Managing Systems*, ser. SEAMS '08. New York, NY, USA: Association for Computing Machinery, 2008, p. 33–40. [Online]. Available: <https://doi.org/10.1145/1370018.1370026>
- [29] C. Henke, M. Tichy, T. Schneider, J. Böcker, and W. Schäfer, "System architecture and risk management for autonomous railway convoys," in *Proceedings of the IEEE International Conference on Systems, Man and Cybernetics (SMC)*, 2008, pp. 1–8.
- [30] R. Alur, "Timed automata," *Theoretical Computer Science*, vol. 126, pp. 183–235, 1999.
- [31] S. Henkler and M. Hirsch, "Contract-based digital twin synthesis for autonomous safety critical systems," in *Proceedings of the 56th Hawaii International Conference on System Sciences*, 2023. [Online]. Available: <https://scholarspace.manoa.hawaii.edu/items/8325ad1a-0df9-4a05-a729-e051c2d5a120>
- [32] B. P. Douglass, *Real-Time Design Patterns: Robust Scalable Architecture for Real-Time Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc., 2002.
- [33] R. Alur, C. Courcoubetis, and D. Dill, "Model-checking in dense real-time," *Information and Computation*, vol. 104, no. 1, pp. 2–34, 1993. [Online]. Available: <https://www.sciencedirect.com/science/article/pii/S0890540183710242>
- [34] P. Bouyer, "An introduction to timed automata," in *Formal Modeling and Analysis of Timed Systems*, ser. Lecture Notes in Computer Science. Springer, 2006, vol. 3829, pp. 1–26.
- [35] C. Weise and D. Lenzkes, "Efficient scaling-invariant checking of timed bisimulation," in *Proc. of STACS'97, LNCS 1200: pages 177–188*. Heidelberg: Springer-Verlag, 1997, pp. 177–188.
- [36] H. Ejersbo Jensen, K. Guldstrand Larsen, and A. Skou, "Scaling up Uppaal," in *Formal Techniques in Real-Time and Fault-Tolerant Systems*, G. Goos, J. Hartmanis, J. Van Leeuwen, and M. Joseph, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2000, vol. 1926, pp. 19–30, series Title: Lecture Notes in Computer Science. [Online]. Available: http://link.springer.com/10.1007/3-540-45352-0_4
- [37] H. Kopetz and W. Steiner, *Real-Time Systems: Design Principles for Distributed Embedded Applications*, 3rd ed. Cham: Springer Cham, 2022, includes 81 b/w illustrations. [Online]. Available: <https://doi.org/10.1007/978-3-031-11992-7>

- [38] STMicroelectronics, *STM32L476xx Ultra-low-power Arm Cortex-M4 32-bit MCU Datasheet*, 11th ed., July 2024. [Online]. Available: <https://www.st.com/resource/en/datasheet/stm32l476je.pdf>
- [39] J. Bengtsson, K. Larsen, F. Larsson, P. Pettersson, and W. Yi, “Up-paal—a tool suite for automatic verification of real-time systems,” in *Proceedings of the DIMACS/SYCON Workshop on Hybrid Systems III: Verification and Control: Verification and Control*. Berlin, Heidelberg: Springer-Verlag, 1996, p. 232–243.
- [40] M. Z. Kwiatkowska, G. Norman, and D. Parker, “Prism 4.0: Verification of probabilistic real-time systems,” in *Proceedings of the 23rd International Conference on Computer Aided Verification (CAV 2011)*, ser. Lecture Notes in Computer Science, vol. 6806. Springer, 2011, pp. 585–591.
- [41] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine, “Kronos: A model-checking tool for real-time systems,” in *Formal Techniques in Real-Time and Fault-Tolerant Systems (FTRFT 1998)*, ser. Lecture Notes in Computer Science, vol. 1486. Springer, 1998, pp. 546–550.

APPENDIX

A. Proof of Theorem 1

Proof. Let $P = |Q_R| \cdot |Q_A|$ be the number of candidate pairs, and let $R(|E|)$ be the cost of computing weak successors $\text{cl}_\tau \circ \text{post}_a \circ \text{cl}_\tau$ over a transition set of size $|E|$ ($R(|E|) = O(|E| \log |E|)$ if DBM is used [34]).

Weak successors $\text{WS}_R(q, a)$ and $\text{WS}_A(q, a)$ depend only on (q, a) , not on the current relation $\mathcal{R}$, hence can be memoised on first use. Total memoisation cost is bounded by $|Q_R| \cdot |\Sigma_{\text{obs}}| \cdot R(|E|_R) + |Q_A| \cdot |\Sigma_{\text{obs}}| \cdot R(|E|_A) = O(R(|E|_R) + R(|E|_A))$ after absorbing alphabet and state factors into R .

The fixed-point loop removes at least one pair per iteration until stabilisation, so the number of iterations is bounded by P . Each iteration performs *Match* on at most P pairs; each *Match* call performs $O(d^2)$ work where d is the maximum out-degree of the symbolic graph since successors are already memoised. In typical TA models d is bounded by the size of the action alphabet, which is small and independent of the state space; we treat it as constant for the asymptotic analysis. Per-iteration work is thus $O(P)$, and total fixed-point work is $O(P^2)$.

Combining: $O(R(|E|_R) + R(|E|_A) + P^2)$. $\square$

B. Proof of Theorem 2

Proof. Termination follows from Theorem 1 and the fact that the main loop of the algorithm only removes pairs from $\mathcal{R}$ until stabilisation.

Let $\Gamma(T_R) = (Q_R, \rightarrow_R)$ and $\Gamma(T_A) = (Q_A, \rightarrow_A)$ denote the zone-based abstractions of the respective TAs and $\mathcal{T}_A$ and $\mathcal{T}_R$ their respective induced TTS.

Standard zone semantics [33], [34] allows us to assume the following properties:

- (Z1) For every concrete state s there is a unique symbolic state $q(s) = (\ell, Z)$ with $s \in q(s)$.
- (Z2) $s \xrightarrow{a, \delta} s'$ in the concrete TTS iff there is a symbolic transition $q(s) \xrightarrow{a} q(s')$ with δ lying in the delay range allowed by $q(s)$.
- (Z3) For any pair of symbolic states q_1, q_2 and any action a , the set of concrete delays $\{\delta \mid \exists s \in q_1, s \xrightarrow{a, \delta} s' \in q_2\}$ is a convex interval with endpoints $\delta^{\min} \leq \delta^{\max}$ (possibly open at either end), computable from the zone q_1 , the

invariant of its location, and the guard of edge a . We write $\delta^{\min}$ and $\delta^{\max}$ for the minimum and maximum delays admitted by the symbolic transition $q_1 \xrightarrow{a} q_2$.

Now we show the two directions of the equivalence.

$(\Leftarrow)$ *Algorithm returns true* $\Rightarrow \mathcal{T}_R \stackrel{rwts}{\sim} \mathcal{T}_A$. Let $\mathcal{R}$ be the final symbolic relation. Define $\stackrel{rwts}{\sim} \subseteq S_R \times S_A$ by $(s_R, s_A) \in \stackrel{rwts}{\sim} \iff (q(s_R), q(s_A)) \in \mathcal{R}$.

We verify each condition of Definition 4.

Condition 1 holds since $\mathcal{R}$ was initialised to $Q_R \times Q_A$ and contains the initial symbolic pair iff the algorithm returned *true*.

For the remaining conditions, the key idea is to lift concrete transitions to the symbolic level, exploit the *Match* check on the symbolic relation, and then use the zone-level delay bounds to find a concrete delay realising the required inequalities. Consider an arbitrary pair (s_R, s_A) with $(q(s_R), q(s_A)) \in \mathcal{R}$. We treat each condition in turn.

Condition 2. Consider a concrete transition $s_R \xrightarrow{a, \delta_R} s'_R$ for some $a \in \Sigma_{\text{out}} \cup \Sigma_{\text{in}}$. By (Z2), this lifts to a symbolic transition $q(s_R) \xrightarrow{a} q(s'_R)$. Since the algorithm accepted $(q(s_R), q(s_A))$, the R-to-A-loop of *Match* (lines 2–10) guarantees the existence of a symbolic transition $q(s_A) \xrightarrow{a} q'_A$ with $(q(s'_R), q'_A) \in \mathcal{R}$. By (Z2)–(Z3), any $\delta_A \in [\delta_A^{\min}, \delta_A^{\max}]$ realises a concrete transition $s_A \xrightarrow{a, \delta_A} s'_A$ with $q(s'_A) = q'_A$; pick any such δ_A . Then $(s'_R, s'_A) \in \stackrel{rwts}{\sim}$ by definition of $\stackrel{rwts}{\sim}$. The argument for transitions initiated from s_A is symmetric, using the A-to-R-loop of *Match* (lines 11–17).

Condition 3. Consider a concrete transition $s_R \xrightarrow{a, \delta_R} s'_R$ with $a \in \Sigma_{\text{out}}$. By (Z2), lift to $q(s_R) \xrightarrow{a} q(s'_R)$. The R-to-A loop of *Match* guarantees a symbolic transition $q(s_A) \xrightarrow{a} q'_A$ with the output-side zone-level inequality $\delta_R^{\max} \leq \delta_A^{\max}$ and $(q(s'_R), q'_A) \in \mathcal{R}$. Since $\delta_R \leq \delta_R^{\max} \leq \delta_A^{\max}$, pick $\delta_A = \max(\delta_R, \delta_A^{\min})$: this lies in $[\delta_A^{\min}, \delta_A^{\max}]$ and satisfies $\delta_A \geq \delta_R$. By (Z2), δ_A realises a concrete transition $s_A \xrightarrow{a, \delta_A} s'_A$ with $q(s'_A) = q'_A$, giving $(s'_R, s'_A) \in \stackrel{rwts}{\sim}$ and $\delta_R \leq \delta_A$ as required.

Condition 4. Consider a concrete transition $s_A \xrightarrow{a, \delta_A} s'_A$ with $a \in \Sigma_{\text{in}}$. By (Z2), lift to $q(s_A) \xrightarrow{a} q(s'_A)$. The A-to-R loop of *Match* guarantees a symbolic transition $q(s_R) \xrightarrow{a} q'_R$ with the input-side zone-level inequality $\delta_R^{\max} \geq \delta_A^{\max}$ and $(q'_R, q(s'_A)) \in \mathcal{R}$. Since $\delta_A \leq \delta_A^{\max} \leq \delta_R^{\max}$, pick $\delta_R = \max(\delta_A, \delta_R^{\min})$: this lies in $[\delta_R^{\min}, \delta_R^{\max}]$ and satisfies $\delta_R \geq \delta_A$. By (Z2), δ_R realises a concrete transition $s_R \xrightarrow{a, \delta_R} s'_R$ with $q(s'_R) = q'_R$, giving $(s'_R, s'_A) \in \stackrel{rwts}{\sim}$ and $\delta_R \geq \delta_A$ as required.

Since each condition of Definition 4 is satisfied for arbitrary $(s_R, s_A) \in \stackrel{rwts}{\sim}$ and the initial pair lies in $\stackrel{rwts}{\sim}$ by Condition 1, the relation $\stackrel{rwts}{\sim}$ is a Relaxed Weak Timed Bisimulation. Hence $\mathcal{T}_R \stackrel{rwts}{\sim} \mathcal{T}_A$.

$(\Rightarrow)$ $\mathcal{T}_R \stackrel{rwts}{\sim} \mathcal{T}_A \Rightarrow \text{algorithm returns true}$. Define the symbolic relation

$$\mathcal{R}_0 = \{(q_R, q_A) \mid \exists s_R \in q_R, s_A \in q_A : (s_R, s_A) \in \stackrel{rwts}{\sim}\}.$$

We show $\mathcal{R}_0$ is closed under *Match*, i.e. for every $(q_R, q_A) \in \mathcal{R}_0$ and every symbolic transition $q_R \xrightarrow{a} q'_R$, there exists q'_A with $q_A \xrightarrow{a} q'_A$, $(q'_R, q'_A) \in \mathcal{R}_0$, and the polarity-correct

zone-level inequality, and vice versa. We proceed by analysing the application of the RWTBS conditions in the loops of the `Match` predicate in Algorithm 1. Specifically, we show that the RWTBS conditions are sufficient to discharge the RWTBS check in the R-to-A-loop (lines 2–10) and the A-to-R-loop (lines 11–17). We provide a detailed argument for the R-loop, as the A-loop is symmetric.

Fix a symbolic transition $q_R \xrightarrow{a} q'_R$; we must exhibit q'_A with $q_A \xrightarrow{a} q'_A$, $(q'_R, q'_A) \in \mathcal{R}_0$, and the polarity-correct zone-level inequality. By definition of $\mathcal{R}_0$, there exist witnesses $s_R \in q_R$ and $s_A \in q_A$ with $(s_R, s_A) \in \sim^{rwts}$. By (Z2) and (Z3), choose a concrete realisation $s_R \xrightarrow{a, \delta_R^*} s'_R$ with $s'_R \in q'_R$. We discharge the RWTBS check by case analysis on the polarity of a .

Case $a \in \Sigma_{\text{out}}$. Pick $\delta_R = \delta_R^{\text{max}}$. Condition 3 yields s'_A, δ_A with $s_A \xrightarrow{a, \delta_A} s'_A$, $(s'_R, s'_A) \in \sim^{rwts}$, and $\delta_R \leq \delta_A$. Combining with $\delta_A \leq \delta_A^{\text{max}}$ from (Z3) gives the zone-level inequality $\delta_R^{\text{max}} \leq \delta_A^{\text{max}}$ required for outputs.

Case $a \in \Sigma_{\text{in}}$. The timing obligation for the input step is discharged by the A-to-R-loop of `Match`. Only the event match is required here, supplied by Condition 2: there exists s'_A with $s_A \xrightarrow{a, \delta_A^*} s'_A$ for some delay δ_A^* and $(s'_R, s'_A) \in \sim^{rwts}$.

In both cases, set $q'_A := q(s'_A)$. By (Z1) and (Z2), the symbolic transition $q_A \xrightarrow{a} q'_A$ exists, and $(q'_R, q'_A) \in \mathcal{R}_0$ by definition of $\mathcal{R}_0$. The R-to-A-loop of `Match` therefore succeeds on (q_R, q_A) .

For the A-to-R loop, the argument is symmetric, with the roles of the two conditions exchanged. Fix a symbolic transition $q_A \xrightarrow{a} q'_A$; we must exhibit q'_R with $q_R \xrightarrow{a} q'_R$, $(q'_R, q'_A) \in \mathcal{R}_0$, and the polarity-correct zone-level inequality. By (Z2) and (Z3), choose a concrete realisation $s_A \xrightarrow{a, \delta_A} s'_A$ with $s'_A \in q'_A$.

Case $a \in \Sigma_{\text{in}}$. Pick $\delta_A = \delta_A^{\text{max}}$. Condition 4 yields s'_R, δ_R with $s_R \xrightarrow{a, \delta_R} s'_R$, $(s'_R, s'_A) \in \sim^{rwts}$, and $\delta_R \geq \delta_A$. Combining with $\delta_R \leq \delta_R^{\text{max}}$ from (Z3) gives the zone-level inequality $\delta_R^{\text{max}} \geq \delta_A^{\text{max}}$ required for inputs.

Case $a \in \Sigma_{\text{out}}$. The A-to-R loop only requires event matching for outputs (lines 11–17); the output-timing constraint is checked in the R-to-A loop (lines 2–10), independently. The event match is supplied by Condition 2: there exists s'_R with $s_R \xrightarrow{a, \delta_R^*} s'_R$ for some δ_R^* and $(s'_R, s'_A) \in \sim^{rwts}$.

In both cases, set $q'_R := q(s'_R)$. By (Z1) and (Z2), the symbolic transition $q_R \xrightarrow{a} q'_R$ exists, and $(q'_R, q'_A) \in \mathcal{R}_0$ by definition of $\mathcal{R}_0$. The A-to-R loop of `Match` therefore succeeds on (q_R, q_A) .

Hence `Match` succeeds on (q_R, q_A) relative to $\mathcal{R}_0$, i.e. $\mathcal{R}_0$ is a post-fixed point of the operator implicitly defined by `Match`. Since Algorithm 1 computes the greatest such post-fixed point, $\mathcal{R}_0 \subseteq \mathcal{R}_{\text{final}}$.

The initial concrete pair $(s_{0,R}, s_{0,A}) \in \sim^{rwts}$ by Condition 1, hence $(q_{R,0}, q_{A,0}) \in \mathcal{R}_0 \subseteq \mathcal{R}_{\text{final}}$. Therefore the algorithm returns `true`.

Combining both directions, we have $\mathcal{T}_R \sim^{rwts} \mathcal{T}_A$ iff the algorithm returns `true`. Therefore, $T_R \sim^{rwts} T_A$ iff the algorithm returns `true`. $\square$

C. Proof of Lemma 1

Proof. Fix $\mu_O \in \Sigma_{\text{out}}$ and let $\sigma' \in \llbracket \mathcal{T}_R \rrbracket$ be an arbitrary trace containing μ_O , induced by a path π' in $\mathcal{T}_R$ of the form $s_R \xrightarrow{\delta'} \left(\xrightarrow{\delta} \middle| \xrightarrow{\mu} \right)^* \xrightarrow{\mu_O} s'_R$ with cumulative delay $\sum^{\mu_O} \delta'$.

By hypothesis, there exists s_A with $(s_R, s_A) \in \sim^{rwts}$. By Condition 3 of Definition 4 applied to π' , there exists a matching path π in $\mathcal{T}_A$ from s_A ending in μ_O with cumulative delay $\sum^{\mu_O} \delta$ satisfying $\sum^{\mu_O} \delta' \leq \sum^{\mu_O} \delta$. The trace σ induced by π lies in $\llbracket \mathcal{T}_A \rrbracket$, so by hypothesis $\sum^{\mu_O} \delta \leq D$. Hence $\sum^{\mu_O} \delta' \leq D$.

Since σ' and the occurrence of μ_O were arbitrary, every output event in every trace of $\mathcal{T}_R$ occurs within D . $\square$

D. Proof of Theorem 3

Proof. Let $\sigma \in \llbracket \mathcal{T}_R \rrbracket$ be arbitrary. If $\sigma \notin A$ then $\sigma \in \llbracket C \rrbracket$ by definition, so assume $\sigma \in A$. By Lemma 1, since $\mathcal{T}_A \models C$, every trace in $\llbracket \mathcal{T}_A \rrbracket$ that satisfies A meets all deadlines of G . Condition 2 of Definition 4 ensures that for every trace $\sigma \in \llbracket \mathcal{T}_R \rrbracket$ satisfying A , there is a trace $\sigma' \in \llbracket \mathcal{T}_A \rrbracket$ satisfying A that matches σ on observable events. Since every such trace of $\mathcal{T}_A$ meets the deadlines of G , so does every trace of $\mathcal{T}_R$ by Lemma 1, including σ . Therefore $\sigma \in G$, hence $\sigma \in \llbracket C \rrbracket$. Since σ was arbitrary, $\llbracket \mathcal{T}_R \rrbracket \subseteq \llbracket C \rrbracket$ and therefore $\mathcal{T}_R \models C$. $\square$

E. Proof of Corollary 1

Proof. Fix an arbitrary $\mu_O \in \Sigma_{\text{out}}(s_R, s_A)$ with deadline D_{μ_O} , and let t_{elapsed} be the time already elapsed from the contract's reference event upon reaching s_R . We show that any input delay $\delta_I \leq D_{\mu_O} - t_{\text{elapsed}} - D(s_R, \mu_O)$ preserves contract satisfaction.

Let $\sigma \in \llbracket \mathcal{T}_R \rrbracket$ be an arbitrary trace passing through s_R at time t_{elapsed} , incorporating an additional input delay δ_I before the next observable event, and reaching an occurrence of μ_O . If $\sigma \notin A$ then $\sigma \in \llbracket C \rrbracket$ by definition of contract satisfaction (Section V), so assume $\sigma \in A$.

By definition of $D(s_R, \mu_O)$, the accumulated delay along σ from s_R to μ_O is at most $D(s_R, \mu_O)$. Adding the input delay δ_I introduced at s_R , the total time at which μ_O occurs in σ is bounded by $t_{\text{elapsed}} + \delta_I + D(s_R, \mu_O)$. By the assumption on δ_I ,

$$\begin{aligned} t_{\text{elapsed}} + \delta_I + D(s_R, \mu_O) &\leq t_{\text{elapsed}} + \\ (D_{\mu_O} - t_{\text{elapsed}} - D(s_R, \mu_O)) + D(s_R, \mu_O) &= D_{\mu_O}. \end{aligned}$$

so μ_O occurs within its deadline in σ . Since μ_O and σ were arbitrary, every output event in every accepting trace of $\mathcal{T}_R$ meets its deadline, hence $\sigma \in G$ and $\sigma \in \llbracket C \rrbracket$. Therefore $\mathcal{T}_R \models C$.

The bound is tight: any $\delta_I > D_{\mu_O} - t_{\text{elapsed}} - D(s_R, \mu_O)$ admits, by definition of $D(s_R, \mu_O)$ as a maximum, a trace in which μ_O occurs at time strictly greater than D_{μ_O} , violating the contract. Taking the minimum over all $\mu_O \in \Sigma_{\text{out}}(s_R, s_A)$ yields the stated bound. $\square$

F. Proof of Theorem 4

Proof. By Theorem 3 applied to C_1 and $\mathcal{T}_{R1}$, we have $\mathcal{T}_{R1} \models C_1$. Similarly, $\mathcal{T}_{R2} \models C_2$. Since C_1 and C_2 are compatible, their composition $C_1 \otimes C_2$ is well-defined. By the soundness of contract composition [8], we have $\mathcal{T}_{R1} \parallel \mathcal{T}_{R2} \models C_1 \otimes C_2$. Therefore, the composed system satisfies the composed contract. $\square$

G. Proof of Theorem 5

Proof. The hypothesis of the theorem strengthens Condition 3 of Definition 4 symmetrically. Unfolded, it requires that for every $(s_R, s_A) \in \overset{rw tbs}{\sim}$ and every $\mu_O \in \Sigma_{out}$, both of the following hold:

- $s_R \xRightarrow{\delta'} (\xRightarrow{\mu_R})^* \xRightarrow{\mu_O} s'_R \implies \exists s'_A : s_A \xRightarrow{\delta} (\xRightarrow{\mu_A})^* \xRightarrow{\mu_O} s'_A \wedge (s'_R, s'_A) \in \overset{rw tbs}{\sim} \wedge \sum_{\mu_O}^{\pi_R} \delta' = \sum_{\mu_O}^{\pi_A} \delta$.
- $s_A \xRightarrow{\delta} (\xRightarrow{\mu_A})^* \xRightarrow{\mu_O} s'_A \implies \exists s'_R : s_R \xRightarrow{\delta'} (\xRightarrow{\mu_R})^* \xRightarrow{\mu_O} s'_R \wedge (s'_R, s'_A) \in \overset{rw tbs}{\sim} \wedge \sum_{\mu_O}^{\pi_R} \delta' = \sum_{\mu_O}^{\pi_A} \delta$.

The first is Condition 3 with equality substituted for $\leq$; the second is its converse. We refer to these below as Condition 3 in the $\mathcal{T}_R \rightarrow \mathcal{T}_A$ direction and in the $\mathcal{T}_A \rightarrow \mathcal{T}_R$ direction, respectively.

Fix $(s_R, s_A) \in \overset{rw tbs}{\sim}$ and $\mu_O \in \Sigma_{out}$.

Forward. Assume $s_R \xRightarrow{\delta_R \mu_O} s'_R$. By Condition 3 in the $\mathcal{T}_R \rightarrow \mathcal{T}_A$ direction under the exact-matching assumption, there exists s'_A with $s_A \xRightarrow{\delta_A \mu_O} s'_A$, $\delta_A = \delta_R$, and $(s'_R, s'_A) \in \overset{rw tbs}{\sim}$.

Backward. Assume $s_A \xRightarrow{\delta_A \mu_O} s'_A$. By Condition 3 in the $\mathcal{T}_A \rightarrow \mathcal{T}_R$ direction under the exact-matching assumption, there exists s'_R with $s_R \xRightarrow{\delta_R \mu_O} s'_R$, $\delta_R = \delta_A$, and $(s'_R, s'_A) \in \overset{rw tbs}{\sim}$.

Both directions match the corresponding observable step with equal elapsed time and a successor pair in $\overset{rw tbs}{\sim}$. Hence $\overset{rw tbs}{\sim}$ satisfies the back-and-forth conditions of a timed weak bisimulation restricted to output events. $\square$