Contract-based Digital Twin Synthesis for Autonomous Safety Critical Systems*

Stefan Henkler
Hamm-Lippstadt University of Applied Sciences
stefan.henkler@hshl.de

Martin Hirsch
University of Applied Sciences and Arts Dortmund
martin.hirsch@fh-dortmund.de

Abstract

The development of Digital Twins is a complex process. The complexity increases with the autonomy and networking of the system under consideration. However, progressive scenarios in the area of smart farming or platooning in road traffic in particular require reliable Digital Twins that can consider safety-critical aspects. Previous approaches do not consider these and are additionally not suitable due to the often manual construction. We present an approach based on a well-defined contract-based engineering that automatically synthesizes the complex interaction with the system under consideration by preserving safety and liveness properties.

Keywords: Digital Twin, Synthesis, Autonomous Systems, Safety Critical Systems, Software Engineering

1. Introduction

Autonomous systems are an important driver of many technical systems from different application areas. Examples can be found in the area of transportation, such as self-driving vehicles or (partially) autonomous vehicles, aircraft or also in agriculture from autonomous combine harvesters to applications in the area of smart farming. What all these systems have in common is that they are strongly networked systems (see Gausemeier et al. (2014)) which have to fulfill safety-critical aspects.

The consideration of safety and liveness aspects is essential, since a misbehavior can lead to a danger for the environment (see e.g. Storey, 1996). This goes hand in hand with the consideration of time and so-called hard real-time systems (Buttazzo, 2011). This means that the non-observance of time constraints leads to faulty behavior. Since a formal verification of at least parts of these systems is essential, the use of Timed Automata (Alur, 1999) and their tool-supported modeling and verification in UPPAAL (Behrmann et al., 2004) has become established or is widely used, especially in the academic environment.

Digital Twins are a promising approach for these systems to significantly and sustainably support the increased complexity given by the high degree of networking of these systems. However, the observation of relevant parameters from these finally distributed systems is highly complex. It must be ensured that relevant events are detected in time, so that it is possible to react to misbehavior in a timely manner. None of the existing approaches address the safety-critical autonomous systems under consideration, nor the use of well-defined engineering paradigms for the development of these systems (see Monteiro et al., 2018; Sreedevi and Santosh Kumar, 2020; Verdouw et al., 2021). In Bano et al. (2022) a synthesis of the Digital Twin behavior is already considered, based on well known process definitions. Since neither safety aspects nor temporal criteria are considered, this approach is not applicable to safety critical systems.

Our approach (see Figure 1) is based on the well-known contract-based paradigm for safety-critical systems (see Sangiovanni-Vincentelli et al., 2012, Broy et al., 2012, Henkler and Eckardt, 2012). A contract consists of an assumption and a guarantee based on this assumption. In terms of behavior, it can be concluded that if an assumed behavior is given, a defined behavior is satisfied by the guarantee. As already shown in Giese et al., 2003, this paradigm can be realized via Timed

*This research is supported by a grant from the Ministry of Economic Affairs, Industry, Climate Action and Energy of the State of North Rhine-Westphalia (MWIDE) as part of the 5G-Landwirtschaft-ML project in the context of the program 5G.NRW (01.05.2022 – 31.12.2024, grant number 005-2108-0039).

URI: https://hdl.handle.net/10125/103453
978-0-9981331-6-4

Automata and a compositional formal verification. Since we are considering safety-critical systems, we can and must assume that this form of well-defined process behavior is given, since otherwise a predictable implementation of the systems would not be possible, which is mandatory.

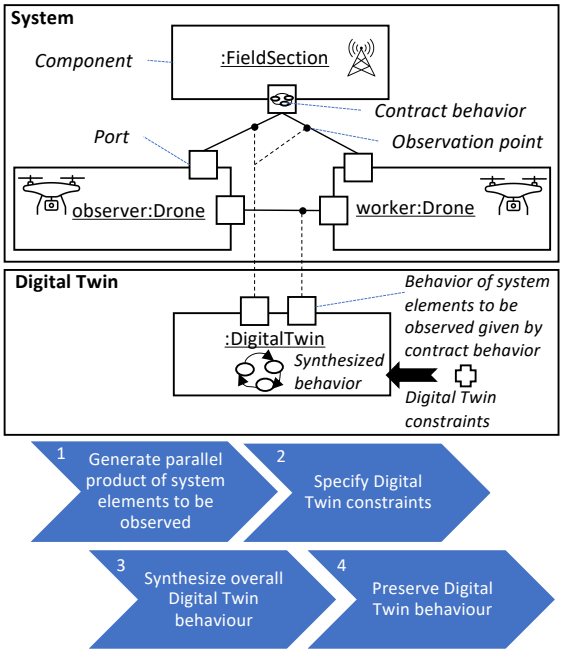

Figure 1. Sketch of approach

Specifically, we assume that these contracts are given in the form of Timed Automata. The behavior to be observed by the Digital Twin results from the composition of the specified contracts, constrained by disallowed states or paths. Accordingly, we call the constraints that the Digital Twin has to consider composition rules. From the known contracts and composition rules, we synthesize the interaction behavior to be observed, which the Digital Twin must monitor. We further ensure that the properties of the contracts are not violated. The associated refinement is the last step of our approach.

As a running example, we use an application from the field of smart farming (see Figure 1). This consists of several drones, in the role of an *observer* and a *worker*, as well as a field section (*:FieldSection*). Further details on the contract behavior are considered in the next sections. The Digital Twin (*:DigitalTwin*) refers to relevant observation points of the *observer* and *worker* drones that are required for a specific use case to be realized. The observation points are linked to the contract behavior of the components under consideration.

In Section 2, we introduce the basic contract-based approach implemented by Timed Automata. We explain our approach to Digital Twin development in Sections 3, 4, and 5. Section 3 describes the 1st and 2nd step of Digital Twin development in terms of the formal description of the composed behavior as well as the definition of constraints based on composition rules. In Section 4 we describe the synthesis and in Section 5 we show how we preserve contract properties from potentially being violated by the synthesis. An evaluation based on a simplified use case from the smart farming domain is presented in Section 6 and we conclude the paper with a summary and outlook in Section 7.

2. Contract-based Design

We assume that the system is designed in a well defined way via contract-based design (for example Sangiovanni-Vincentelli et al., 2012). For the formal, model-based specification we assume to use Time Automota as defined in the following.

Definition 2.1 (Timed Automaton) *A* Timed Automaton A *is a tuple* $(L, l^0, \Sigma, C, I, T)$ *where*

- L *is the set of locations*
- $l^0 \in L$ *is the initial location,*
- Σ *is the finite set of events where the symbol* τ *is used for internal events (silent transitions),*
- $I : L \rightarrow \Phi_{dc}(C)$ *assigns each location a location invariant as a downwards closed clock constraint,*
- C *is the finite set of clocks, and*
- $T \subseteq L \times \Sigma \times \Phi(C) \times 2^C \times L$ *is the finite set of transitions* $t = (l, e, g, r, l') \in T$ *with*
  - $l \in L$ *the source location,*
  - $e \in \Sigma$ *the related event,*
  - $g \in \Phi(C)$ *the time guard as a general clock constraint,*
  - $r \subseteq C$ *a set of clocks to be reset, and*
  - $l' \in L$ *the target location.*

As a basis, we therefore assume that the contracts, i.e., the assumptions and guarantees, are described in the form of Timed Automata. An excerpt of such behavior from the Smart Farming application domain is shown in Figure 2.

The example shows how the process of task assignment and fulfillment of a drone can be described via a Timed Automata contract. Initially the work is started via *initWork!*. In the state *work* we send every 50

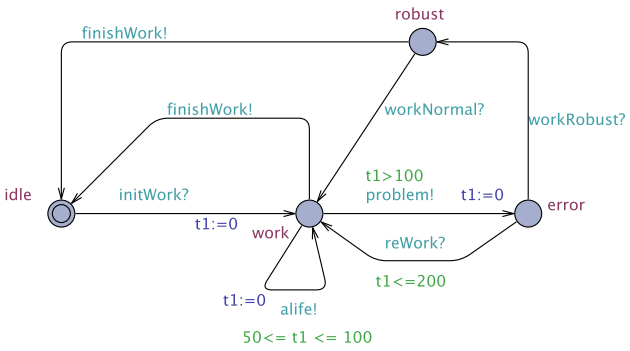

Figure 2. Part of Drone Worker Contract (Timed Automaton)

to 100 time units a *alife!* message. In case of a present problem, it will switch to the *error* state. This can be exited back to the *work* state within 200 time units and the receipt of a *rework* message. Alternatively, it can be changed to the *robust* state after any time and the receipt of a *workRobust* message. From the *robust* and *work* state it is possible to switch to the *idle* state when the work is done.

We assume further that the correctness of the contracts, like deadlock freedom, are formerly verified via TCTL properties as supported in UPPAAL (see Behrmann et al., 2004).

3. Constraining Digital Twin Contracts

The first step of the development of the Digital Twin behavior based on contracts consists of the description of the composed behavior of the systems to be observed and the description of the dependencies of these systems.

Therefore, the possible overall behavior of the Digital Twin is based on the definition in 3.1, which results from the parallel behavior of the systems to be observed.

Definition 3.1 (Parallel Composition) *Let* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ *and* $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$ *be two timed automata with* $C_1 \cap C_2 = \varnothing$ *and* $\Sigma_1 \cap \Sigma_2 = \varnothing$. *We define the parallel composition* $A_1 \parallel A_2$ *as a product automaton* $A_P = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$, *where*

- $L_P = L_1 \times L_2$,
- $l_P^0 = (l_1^0, l_2^0)$,
- $\Sigma_P = \Sigma_1 \cup \Sigma_2$,
- $I_P : L_P \rightarrow \Phi(C_1) \cup \Phi(C_2)$ *with* $I_P((l_1, l_2)) = I_1(l_1) \wedge I_2(l_2)$,
- $C_P = C_1 \cup C_2$,
- $T_P \subseteq L_P \times \Sigma_P \times \Phi(C_P) \times 2^{C_P} \times L_P$, *with*
  - $((l_1, l_2), e_1, g_1, r_1, (l_1{}', l_2)) \in T_P \Leftrightarrow (l_1, e_1, g_1, r_1, l_1{}') \in T_1$, *and*
  - $((l_1, l_2), e_2, g_2, r_2, (l_1, l_2{}')) \in T_P \Leftrightarrow (l_2, e_2, g_2, r_2, l_2{}') \in T_2$.

In accordance with Bengtsson and Yi, 2003; Henzinger et al., 1992, we do only allow to model upper bounds for location invariants. Lower bounds can be expressed by using time guards at the incoming transitions of a location. Consequently, restricting location invariants to downwards closed clock constraints does not change the expressiveness of timed automata, but simplifies modeling and semantical analysis.

We consider the specification of system properties in terms of safety and liveness for a given behavioral specification Henzinger, 1992; Lamport, 1977. *Safety properties* state that something bad will never happen during the execution of a program. *Liveness properties* state that something good will happen eventually.

Transferring these properties to the composition rules proposed in this paper, we are able to specify both safety and liveness properties. Safety properties can be specified (1) by means of *state composition rules* in terms of forbidden state combinations of the parallel execution and (2) by means of *event composition automata* by adding further time constraints to time guards of selected transitions. Liveness properties in turn can be specified through state composition rules and event composition automata by adding further time constraints to location invariants of location combinations of the parallel execution.

3.1. State Composition Rules

When a Digital Twin observes more than one contract, certain combinations of states within their behavioral models might be forbidden due to given system requirements. Consequently, we need a formalism to define the restricted state combinations of the affected timed automata. This formalism is described and defined formally in the following by the notion of *state composition rules*, which is based on the syntax and semantics of the given contract formalism.

Syntactically, a state composition rule consists of a set location predicates connected by the Boolean relations meet and join, all together surrounded by a negation. A location predicate specifies a location in combination with a set of clock valuations, while this combination is not permitted in relation with other location predicates specified in that rule. The clock

valuations of a location predicate are specified by clock constraints.

First, we define the form of clock constraints allowed for location predicates. We do this by the type of *upwards closed clock constraints* in the following.

Definition 3.2 (Upwards Closed Clock Constraint) *For a set C of clocks, the set $\Phi_{uc}(C) \subset \Phi(C)$ of* upwards closed clock constraints *is inductively defined by the grammar*

$$\varphi ::= x \sim n \mid x - y \sim n \mid \varphi \wedge \varphi \mid true,$$

where $x, y \in C$, $\sim \in \{\geq, >\}$, $n \in \mathbb{N}$.

Upwards closed clock constraints only allow to specify clock constraints or conjunctions of clock constraints which describe lower bounds for a clock or the difference of two clocks. In the context of state composition rules this is used to describe the lower bound of the forbidden time interval for a location. Upper bounds are not permitted, as their evaluation might result in a location invariant for a location which is not downwards closed.

Using the definition of upwards closed clock constraints, we can give a proper formal definition of the syntax of *location predicates* in the following.

Definition 3.3 (Location Predicate) *For a timed automaton $A = (L, l^0, \Sigma, C, I, T)$, a location $l \in L$ and an upwards closed clock constraint $\varphi \in \Phi_{uc}(C)$ the set $\Gamma(A)$ of location predicates $\gamma = (l, \varphi)$ is defined by*

$$\Gamma(A) = L \times \Phi_{uc}(C).$$

A location predicate is the combination of a location and downwards closed clock constraint for a timed automaton A, used to define a forbidden set of clock valuations φ for a timed automaton location l.

We finally define the syntax of *state composition rules*.

Definition 3.4 (State Composition Rule) *For two timed automata $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ and $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$ the set $R^S(A_1, A_2)$ of* state composition rules *ρ is defined by the grammar*

$$\begin{aligned} \rho &::= \neg\rho_\gamma \\ \rho_\gamma &::= \rho_\gamma \wedge \rho_\gamma \mid \rho_\gamma \vee \rho_\gamma \mid \gamma \end{aligned}$$

where $\gamma \in \Gamma(A_1) \cup \Gamma(A_2)$.

The first part of the grammar defines that each state composition rule is surrounded by a negation, in order to emphasize the fact that it specifies a forbidden combination of location predicates for the parallel execution of the given contract behaviour. The second part specifies the possibility of using the Boolean operators meet and join in order to express which combination of location predicates is forbidden.

3.2. Event Composition Automata

The formal definition of event composition automata is given in Definition 3.5.

Definition 3.5 (Event Composition Automaton) *Let $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ and $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$ be two timed automata. An* event composition automaton *$A_E \in R^A(A_1, A_2)$ is again a timed automaton as a tuple $(L_E, l_E^0, \Sigma_E, C_E, I_E, T_E)$, where*

- *L_E is a finite non empty set of locations,*
- *$l_E^0 \subseteq L$ is the initial location,*
- *$\Sigma_E \subseteq \Sigma_1 \cup \Sigma_2$ is the finite set of events to be observed,*
- *$I : L \rightarrow \Phi_{dc}(C_E)$ assigns each location a downwards closed clock constraint,*
- *C_E is a finite set of clocks, with $C_E \cap (C_1 \cup C_2) = \varnothing$*
- *$T_E \subseteq L_E \times \Sigma_E \times \Phi(C_E) \times 2^{C_E} \times L_E$ is a finite set of transitions $t = (l, e, g, r, l') \in T_E$, with*
  - *$l \in L_E$ is the source location,*
  - *$e \in \Sigma_E$ is the observed event,*
  - *$g \in \Phi(C_E)$ is the time guard,*
  - *$r \subseteq C_E$ is a set of clocks to be reset, and*
  - *$l' \in L_E$ is the target location.*

For two timed automata A_1 and A_2, we define the syntax of an event composition automaton as a timed automaton (see Definition 2.1), which only uses events from A_1 and A_2 and whose set of clocks is disjoint to the set of clocks of A_1 and to the set of clocks of A_2.

Semantically, an event composition automaton only observes event occurrences of the given contract automata. Consequently, only those events can be used in an event composition automaton, as others can never be observed. Additionally, the set of clocks of the event composition automaton is restricted to be disjoint set of clocks of the contract automata. This way, it is guaranteed that the event composition automaton cannot widen the time intervals of event sequences of the automata to be synchronized. If the event composition automaton could widen those time intervals, it might happen that earlier verified deadlines of the contract automata cannot be met

anymore. Consequently, verification results could not be guaranteed to be preserved after performing the synthesis.

4. Synthesis of the Digital Twin Behavior

As we defined the input for the synthesis algorithm, which are the composition rules and the contract automata in the preceding sections, we can proceed with the specification of the synthesis algorithm for the Digital Twin behavior itself. For reasons of simplification we only give examples and definitions for the procedure based on two contract automata. The concept can be easily transfered to an arbitrary number of contract automata, by extending the definitions concerning the two input timed automata to sets of timed automata.

4.1. Applying State Composition Rules

In the preceding section, we defined the parallel composition of contract automata (see Definition 3.1) as an explicit model for the parallel execution of those. State composition rules, as defined in section 3.1, specify both safety and liveness properties for this parallel execution. Accordingly, the parallelly composed timed automaton has to be modified according to the specified state composition rules. This way, an explicit model for the parallel execution of role automata is obtained, which also considers the specified safety and liveness properties of state composition rules.

For a given parallelly composed location l and a given state composition rule r, the first step is to evaluate the location predicates of r in order to find out if the location invariant of the location l is affected by the rule r. This is defined by the *location predicate evaluation* below.

Definition 4.1 (Location Predicate Evaluation) *Given two timed automata* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$, $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$, *their parallelly composition* $A_P = A_1 \parallel A_2 = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$, *a corresponding parallel composed location* $l_p = (l_1, l_2)$, *and a location predicate* $\gamma = (l, \varphi)$ *with* $l \in L_1 \cup L_2$ *and* $\varphi \in \Phi_{uc}(C_1) \cup \Phi_{uc}(C_2)$ *the* location predicate evaluation *is a function* $\gamma : L_P \to \Phi_{uc}(C_P) \cup \{false\}$ *defined with*

$$\gamma(l_p) = \begin{cases} \varphi, & \text{iff } (l = l_1) \vee (l = l_2), \\ false, & else. \end{cases}$$

The *location predicate evaluation* returns the forbidden clock valuations in form of a clock constraint for a parallelly composed location (l_1, l_2) and a given location predicate (l, φ). If one of the locations referred to in the composed location is equal l, the forbidden set of clock valuations is exactly described by φ. If none of those locations is equal to l, permitting the clock valuations described by φ is not applicable, and therefore the set of forbidden clock valuations is described by $false$. This means that for the location (l_1, l_2) no clock valuation is restricted by the location predicate (l, φ).

The location predicate evaluation is used as part of each *state composition rule evaluation* in order to return the permitted set of clock valuations for each composed location (l_1, l_2). The state composition rule evaluation is defined as follows.

Definition 4.2 (State Composition Rule Evaluation) *Given two timed automata* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$, $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$, *their parallel composition* $A_P = A_1 \parallel A_2 = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$, *a corresponding parallelly composed location* $l_p = (l_1, l_2)$, *and a state composition rule* $\rho \in R^S(A_1, A_2)$. *The* state composition rule evaluation *is a function* $\rho : L_P \to \Phi_{dc}(C_P) \cup \{false\}$ *defined with*

$$\rho(l_p) = \begin{cases} \neg\rho_1(l_p), & \text{iff } \rho \text{ is of the form } \neg\rho_1, \\ \rho_1(l_p) \wedge \rho_2(l_p), & \text{iff } \rho_\gamma \text{ is of the form } \rho_1 \wedge \rho_2, \\ \rho_1(l_p) \vee \rho_2(l_p), & \text{iff } \rho_\gamma \text{ is of the form } \rho_1 \vee \rho_2, \\ \gamma(l_p), & \text{iff } \rho_\gamma \text{ is the literal } \gamma. \end{cases}$$

where $\gamma \in \Gamma(A_1) \cup \Gamma(A_2)$.

Each state composition rule consists of the negation of location predicates or conjunctions and disjunctions of location predicates. Accordingly, all these forms have to be considered within the evaluation of a state composition rule.

The negated form of a state composition rule is simply evaluated by negating the result of the unnegated rule by the rules of Boolean algebra. Note that the negation of an upwards closed clock constraint is obtained by inverting the relational operators "$>$" to $\leq$ and "$\geq$" to "$<$". The conjunctive and disjunctive forms of state composition rules are in turn evaluated by applying the corresponding Boolean operators on the evaluations of those rules. Finally, the evaluation of an atomic location predicate is evaluated by means of the location predicate evaluation as defined above (Definition 4.1).

On the basis of the definitions of the location predicate evaluation and the state composition rule evaluation we proceed with the definition of the *state composition conform* timed automaton, which

is the resulting automaton after a given set of state composition rules have been applied.

Definition 4.3 (State Composition Conformance) *Let $A_P = A_1 \parallel A_2 = (L_P, l_P^0, \Sigma_P, C_P, I_P, T_P)$ be the parallel composition of the timed automata A_1 and A_2. Further let $R_1^S \subseteq R^S(A_1, A_2)$ be a set of state composition rules specified over A_1 and A_2. The* state composition conform*, parallelly composed timed automaton $A_{SC} = (L_{SC}, l_{SC}^0, \Sigma_{SC}, C_{SC}, I_{SC}, T_{SC})$ is defined with*

- $L_{SC} = L_P \setminus L_R$, *where* $L_R = \{l_p \mid l_p \in L_P$ *and* $\forall\ \rho_1, \ldots, \rho_n \in R_1^S : I(l_p) \wedge \rho_1(l_p) \wedge \ldots \wedge \rho_n(l_p) = false\}$,
- $l_{SC}^0 = l_P^0 \Leftrightarrow l_P^0 \in L_{SC}$,
- $\Sigma_{SC} = \Sigma_P$,
- $I_{SC} : L_{SC} \rightarrow \Phi(C_{SC})$ *with* $I_{SC}(l_p) = I_P(l_p) \wedge \rho_1(l_p) \wedge \ldots \wedge \rho_n(l_p), \forall\ \rho_1, \ldots, \rho_n \in R_1^S$,
- $C_{SC} = C_P$,
- $T_{SC} \subseteq L_{SC} \times \Sigma_{SC} \times \Phi(C_{SC}) \times 2^{C_{SC}} \times L_{SC}$, *with* $(l_p, e, g, r, {l_p}') \in T_{SC} \Leftrightarrow (l_p, e, g, r, {l_p}') \in T_P \wedge l_p, {l_p}' \in L_{SC}$.

For a given set of state composition rules R_1^S, the state composition conform timed automaton is defined as the timed automaton where each of the composition rules $\rho \in R_1^S$ has been applied to each of the locations $l_P \in L_P$.

The set of state composition conform locations L_{SC} is the original set of locations L_P, without those locations in L_R which are restricted by the state composition rules. The restricted locations L_R are those, whose invariant $I_P(l_p)$ in conjunction with each of the state composition rule evaluations $\rho_n(l_p)$ is $false$.

The initial location l_{SC}^0 of the state composition conform is the same as the one of the parallelly composed timed automaton, as long as it is not removed by the appliance of the state composition rules. The set of events and the set of clocks stays the same.

The invariant of each location $I_{SC}(l_p)$ is defined as the conjunction of the original invariant $I_P(l_p)$ with each of the state composition rules' evaluations $\rho_n(l_p)$. Note at this point, that the invariant does not change for state composition rules that are not applicable for the current location. The reason for this is that the evaluation in this case simply results in the clock constraint $true$ (see Definition 4.1), which is the identity element for conjunction in Boolean algebra.

The set of transitions T_{SC} of the state composition conform timed automaton is the set of transitions T_P of the parallel composition without those transitions which are incoming or outgoing transitions of restricted locations.

4.2. Applying Event Composition Automata

Event composition automata specify additional synchronization behavior for this parallel composition by means of a special type of timed automata (see section 3.2). Accordingly, event composition automata also have to be applied to the explicit model of the parallel execution of the contract automata, which is the parallelly composed timed automaton. This forms the next step in the synthesis algorithm, which implies that state composition rules (if specified) have already been applied to the parallel composition. Consequently, in this section we describe how event composition automata are applied to a parallelly composed, state composition conform timed automaton.

Similar to the parallel composition used for the parallel execution of the contract automata, applying event composition automata can also be compared to the parallel composition operator of the process algebra *Calculus of Communicating Systems (CCS)* Milner, 1989 or the *networks of timed automata* formalism defined in Yi et al., 1994. The fundamental difference is that for the event composition automaton appliance only synchronization of events is taken into account, as event composition automata do not define any new event occurrences for the parallel execution. Consequently, the result is a product automaton where the locations of the parallel composition are multiplied with the locations of the event composition automaton. Furthermore, the transitions of the event composition automaton are synchronized with the transitions of the parallel composition, although they do not take the channel concept into account. This means that a synchronization between a parallelly composed transition and an event composition automaton transition does not change the event, as the event composition automaton only observes the event occurrences of the parallel execution. This type of synchronization can also be denoted as *silent*, as it does not change the external behavior of the parallel composition.

What the appliance can change is (1) the set of clock resets of synchronized transitions, (2) the time guards of synchronized transitions, and (3) the location invariants of corresponding parallelly composed locations. All of this is only possible in the scope of the set of clocks of the event composition automaton, which has to be disjoint to the set of clocks of the parallelly composed timed automaton. This way, it is achieved that the additional constraints do not widen the time intervals

specified within the parallel composition, which then again guarantees that earlier verified deadlines are still met after applying the event composition automata. The addition of further time constraints is described in detail below.

The formal definition of the *event composition conform* timed automaton is given by definition 4.4, which is the state composition conform, parallelly composed timed automaton to which an event composition automaton has been applied.

Definition 4.4 (Event Composition Conformance) *Let* $A_{SC} = (L_{SC}, l^0_{SC}, \Sigma_{SC}, C_{SC}, I_{SC}, T_{SC})$ *be a state composition conform, parallelly composed timed automaton originating from the timed automata* $A_1 = (L_1, l^0_1, \Sigma_1, C_1, I_1, T_1)$ *and* $A_2 = (L_2, l^0_2, \Sigma_2, C_2, I_2, T_2)$ *with* $C_1 \cap C_2 = \varnothing$ *and* $\Sigma_1 \cap \Sigma_2 = \varnothing$. *Furthermore, let* $A_E = (L_E, l^0_E, \Sigma_E, C_E, I_E, T_E) \in R^A(A_1, A_2)$ *be an event composition automaton for* A_1 *and* A_2. *We define the* event composition conform *and state composition conform, parallelly composed timed automaton* $A_{EC} = (L_{EC}, l^0_{EC}, \Sigma_{EC}, C_{EC}, I_{EC}, T_{EC})$ *with*

- $L_{EC} \subseteq L_1 \times L_2 \times L_E$, *with* $(l_1, l_2, l_e) \in L_{EC}$ *iff* $(l_1, l_2) \in L_{SC}$ *and* $I_{SC}((l_1, l_2)) \wedge I_E(l_e) \neq false$) *and* (l_1, l_2, l_e) *is reachable trough* T_{EC},
- $l^0_{EC} = (l^0_1, l^0_2, l^0_e)$, *iff* $(l^0_1, l^0_2, l^0_e) \in L_{EC}$,
- $\Sigma_{EC} = \Sigma_1 \cup \Sigma_2$,
- $I_{EC} : L_{EC} \rightarrow \Phi(C_1) \cup \Phi(C_2) \cup \Phi(C_E)$ *with* $I_{EC}((l_1, l_2, l_e)) = I_{SC}((l_1, l_2)) \wedge I_E(l_e)$,
- $C_{EC} = C_1 \cup C_2 \cup C_E$,
- $T_{EC} \subseteq L_{EC} \times \Sigma_{EC} \times \Phi(C_{EC}) \times 2^{C_{EC}} \times L_{EC}$, *with*
  - $((l_1, l_2, l_e), e_1, g_1, r_1, (l_1{}', l_2, l_e)) \in T_{EC} \Leftrightarrow$ $((l_1, l_2), e_1, g_1, r_1, (l_1{}', l_2)) \in T_{SC} \wedge$ $\forall\, l_e{}' \in L_E : (l_e, e_1, g_e, r_e, l_e{}') \notin T_E$,
  - $((l_1, l_2, l_e), e_2, g_2, r_2, (l_1, l_2{}', l_e)) \in T_{EC} \Leftrightarrow$ $((l_1, l_2), e_2, g_2, r_2, (l_1, l_2{}')) \in T_{SC} \wedge$ $\forall\, l_e{}' \in L_E : (l_e, e_2, g_e, r_e, l_e{}') \notin T_E$,
  - $((l_1, l_2, l_e), e_1, g_1 \wedge g_e, r_1 \cup r_e, (l_1{}', l_2, l_e{}')) \in T_{EC} \Leftrightarrow$ $((l_1, l_2), e_1, g_1, r_1, (l_1{}', l_2)) \in T_{SC} \wedge$ $(l_e, e_1, g_e, r_e, l_e{}') \in T_E$,
  - $((l_1, l_2, l_e), e_1, g_1 \wedge g_e, r_1 \cup r_e, (l_1, l_2{}', l_e{}')) \in T_{EC} \Leftrightarrow$ $((l_1, l_2), e_2, g_2, r_2, (l_1, l_2{}') \in T_{SC} \wedge$ $(l_e, e_2, g_e, r_e, l_e{}') \in T_E$.

The event composition conform automaton forms the explicit model for the parallel execution of the contract automata, while also respecting the specified state composition rules and event composition automaton.

The locations of the event composition conform automaton L_{EC} are a subset of the cross product $L_1 \times L_2 \times L_E$, including only those locations which are (1) state composition conform, (2) whose invariant $I_{EC}((l_1, l_2, l_e)) = I_{SC}((l_1, l_2)) \wedge I_E(l_e)$ is not equal to $false$, and (3) which is in fact reachable trough the transitions T_{EC} of this automaton.

Equal to the definition of state composition conformance (Definition 4.3), the invariant of each event composition conform location (l_1, l_2, l_e) is constructed by the conjunction of the invariant of the state composition conform location $I_{SC}(l_1, l_2)$ and the invariant of the event composition automaton location $I_E(l_e)$.

The definition of the set of transitions T_{EC} of A_{EC} differentiates between four cases. The first two cases describe the transitions which are not synchronized between A_{SC} and A_E. Here it is further distinguished that it is either a transition where A_1 changes its location or where A_2 changes it location. In both cases there is no corresponding transition in T_E which could be synchronized. The last two cases describe the transitions which are synchronized between A_{SC} and A_E. Here, it is also further distinguished if A_1 changes its location or A_2.

Composed transitions which refer to a transition $t_e \in T_E$ of the event composition automaton but not to a transition $t_{sc} \in T_{SC}$ of the state composition conform automaton are not considered, as t_e can in this case not be synchronized with a transition of the state composition conform automaton. Note that these kind of transitions might actually exist, although the event composition automaton is defined to only listen to those events, which are either in A_1 or in A_2. Some event occurrences might be removed by the state composition conformance, though. This cannot be taken into account for the specification of event composition automata, as this takes place earlier than the state composition rule appliance from a developer's point of view.

For reasons of simplification, we refer to a state composition conform and event composition conform, parallelly composed timed automaton simply as *composition conform*.

5. Preserving Digital Twin Behavior

The last step of our approach to develop the Digital Twin behavior ensures that the contract properties have

not been violated by the synthesis. The defined *Observational Timed Bisimulation* (see Definition 5.1) is a suitable equivalence relation which has to hold between the original parallel composition and the composition conform timed automaton.

Definition 5.1 (Observational Timed Bisimulation) *Given two timed automata* $A_1 = (L_1, l_1^0, \Sigma_1, C_1, I_1, T_1)$ *and* $A_2 = (L_2, l_2^0, \Sigma_2, C_2, I_2, T_2)$*, with* $C_1 \subseteq C_2$*,* $\Sigma_2 \subseteq \Sigma_1$*. Furthermore, let* $S_{A_1} = (Q_{A_1}, q_1^0, \Sigma_1, T_{\delta_1}, T_{\Sigma_1})$ *and* $S_{A_2} = (Q_{A_2}, q_2^0, \Sigma_2, T_{\delta_2}, T_{\Sigma_2})$ *be their corresponding timed transition systems.* A_1 *and* A_2 *are observational bisimulation equivalent, denoted* $A_2 \preceq A_1$ *iff an* observational timed bisimulation *relation* $\Omega \subseteq Q_{A_2} \times Q_{A_1}$ *exists, with* $(q_2^0, q_1^0) \in \Omega$ *and* $\forall\, ((l_2, \nu_2), (l_1, \nu_1)) \in \Omega$ *the following properties hold:*

$$\begin{aligned} &\forall\, ((l_2,\nu_2), e, (l_2',\nu_2')) \in T_{\Sigma_2} \\ &\exists\, ((l_1,\nu_1), e, (l_1',\nu_1')) \in T_{\Sigma_1} \\ \wedge \forall c \in C_1 : \nu_2'(c) = \nu_1'(c) &: ((l_2',\nu_2'), (l_1',\nu_1')) \in \Omega, \end{aligned} \tag{1}$$

$$\begin{aligned} &\forall\, ((l_2,\nu_2), (l_2,\nu_2+\delta)) \in T_{\delta_2} \\ &\exists\, ((l_1,\nu_1), (l_1,\nu_1+\delta)) \in T_{\delta_1}, \\ \delta \in R_+ : ((l_2',\nu_2+\delta), &(l_1',\nu_1+\delta)) \in \Omega, \end{aligned} \tag{2}$$

$$\begin{aligned} &\forall\, ((l_1,\nu_1), e?, (l_1',\nu_1')) \in T_{\Sigma_1} \\ \exists\, ((l_2,\nu_2), e?, (l_2',\nu_2')) &\in T_\tau(S_{A_2}) \wedge ((\nu_1 \neq \nu_1') \\ \Rightarrow (\forall c \in C_1, \nu_1'(c) = 0 : \nu_2'(c) &= 0)) : \\ &((l_1',\nu_1'), (l_2',\nu_2')) \in \Omega, \end{aligned} \tag{3}$$

$$\begin{aligned} &\forall\, ((l_1,\nu_1), e!, (l_1',\nu_1')) \in T_{\Sigma_1} \\ (\, &\exists\, ((l_2,\nu_2), e!, (l_2',\nu_2')) \in T_\tau(S_{A_2}) \vee \\ &\exists ((l_1,\nu_{1-}), e!, (l_1',\nu_{1-}')) \in T_{\Sigma_1} \\ &\exists\, ((l_2'',\nu_2''), e!, (l_2',\nu_2')) \in T_\tau(S_{A_2}), \\ \forall c \in C_1 : \nu_{1-}(c) < \nu_1(c)\,) &\wedge ((\nu_1 \neq \nu_1') \\ \Rightarrow (\forall c \in C_1, \nu_1'(c) = 0 : \nu_2'(c) &= 0)) : \\ &((l_2',\nu_2'), (l_1',\nu_1')) \in \Omega \end{aligned} \tag{4}$$

Additionally, the following property has to hold for A_2*:* $\forall\, s_2 \in Q_{A_2}\ (\exists\, (s_2, e, s_2') \in T_{\Sigma_2} \vee \exists\, (s_2, s_2') \in T_{\delta_2})$.

The first two properties for states $((l_2, \nu_2), (l_1, \nu_1))$ of the relation correspond to the definition of timed simulation. The third property realizes that for each event transition of the more abstract automaton with a receiving event, there exists a transitive event transition in the more concrete automaton. This implies that there also has to be a corresponding receiving event transition in the concrete system for the last possible point in time. If this was not the case, this property would be violated. To exemplify this, assume a transition $t_1 = ((l_1, \nu_1), e?, (l_1', \nu_1')) \in T_{\Sigma_1}$ where the clock valuation ν_1 represents the last possible point in time. For this transition t_1 there also has to exist a corresponding transition t_2 in $T_\tau(S_{A_2})$, and as all delay transitions of S_{A_2} also have to be in S_{A_1} (due to the second property), the transition t_2 has to be provided at the same point in time. Consequently, with this property we guarantee that no event is lost in a communication channel for the reason that A_2 stops listening to the channel too early.

The fourth property is defined similarly, but additionally it is considered that there might already exist an event transition in S_{A_1} earlier in time, for which there exists a transitive transition in S_{A_2}. In this case, the target states of the earlier transitive transition $((l_2'', \nu_2''), e!, (l_2', \nu_2'))$ of S_{A_2} has to be in the simulation relation together with (l_1', ν_1').

Furthermore, observe that for the third and the fourth property, the clock resets are also regarded by comparing the clock valuations ν_1 and ν_1' of the transition $((l_1, \nu_1), e, (l_1', \nu_1'))$ with each other. If they differ, all the clocks which are equal to zero in ν_1' also have to be equal to zero in ν_2' in $((l_2, \nu_2), e, (l_2', \nu_2')) \in T_\tau(S_{A_2})$, to ensure that no clock reset is lost in the concrete time automaton.

With the observational timed bisimulation defined above, we have a suitable equivalence relation, which, if established, preserves the relevant behavior in terms of TCTL formulas.

6. Evaluation

In this section, we will perform an evaluation of the approach using a simplified use case from the smart farming domain. We simplify the contract presented in Section 2 and reduce the state space and consider the following two contracts implemented by a drone.

In Figure 3 we consider a simplified contract to complete a task and in Figure 4 we consider the registration of a drone on a field or a section of a field.

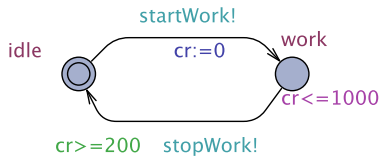

Figure 3. Simplified Drone Worker Contract

The simplified work contract can now immediately start the work by sending a *startWork!* message. It can at most rest in work for 1000 time units but must stay in this location for at least 200 time units. A simplified version of a registree contract automaton allows the drone to immediately send a *register!* message. The

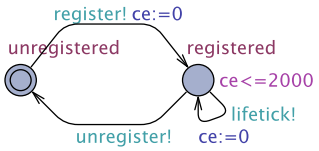

Figure 4. Simplified Drone Registration Contract

data reception mechanism is given by a simple lifetick which has to be called at least every 2000 time units.

The fully obserable behavior is given by the parallel production of both contract automaton in Figure 5. In this product automaton, each location of the work contract automaton is combined with each location of the registree contract automaton. Accordingly, the location invariant of the composed location is the conjunction of each corresponding single invariant. The set of incoming and outgoing transitions of a composed location is the union of the incoming and outgoing transitions of each corresponding single location. This way, an interleaving of events is achieved, as an event of the work contract automaton and an event of the registree contract automaton can never occur at the same transition. A parallel execution of two distinct events is still possible, though, as the transitions are executed in zero-time and time does not have to pass in locations.

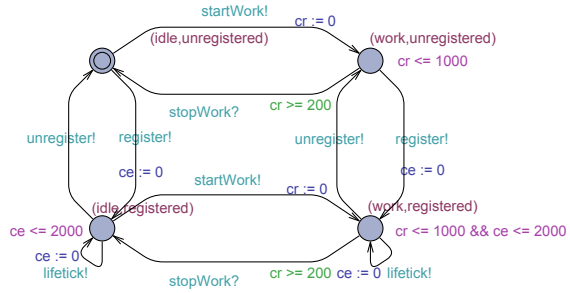

Figure 5. Parallely Composed Contracts

The restriction the Digital Twin has to consider is given by the state composition rule dt_1:

$$dt_1 \quad = \quad \neg((unregistered, true) \wedge (work, true)).$$

Meaning, it is not allowed that the drone is in *work* state and not registered (on the field).

We examine $dt_1 = \neg((unregistered, true) \wedge (work, true))$, which specifies that for any clock valuation the combination of locations *unregistered* and *work* is not permitted for the parallel execution. Consequently, we search the locations of the parallelly composed automaton, and remove all locations from this automaton, which refer to the single locations *unregistered* and *work*. As the example presented here is constructed only of the two timed automata of the work contract and the registree contract, the only location to be removed is the location *(work,unregistered)*. The resulting automaton is depicted in Figure 6.

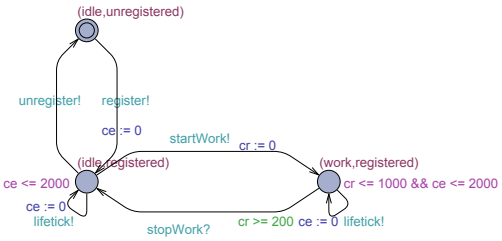

Figure 6. Synthesized Digital Twin (applied Rule r1)

After the composition rule has been applied to the parallelly composed timed automaton, it is not ensured anymore that the visible behavior of each of the particular contract automata is still preserved as discussed in Section 5. In the considered use case we can simply show that this is fulfilled. Despite the fact that the state *work, unregistered* has been removed, the messages lying on the path are offered. I.e. for the message *startWork!* which can be sent at some point directly on the path from *idle, unregistered* to *work, unregistered*, another path exists which has the same properties. This path is given by the transition between the states *idle, registered* to *work, registered*. With respect to the external visible behavior to be supported, it doesn't matter that this requires sending or receiving another event as proposed by the refinement relation in Section 5.

7. Summary and Outlook

In this paper, we have presented an approach to automatically synthesize the behavior of Digital Twins. Our approach is based on the formal definition of contracts implemented by Timed Automata and takes into account safety and liveness properties as required for the domain of safety-critical autonomous applications under consideration. Adaptation of contracts at runtime to respond to changing, unknown environments is, besides the considered networking, an important challenge of autonomous systems in particular (see Hirsch et al., 2008). This is not yet addressed by the current approach and represents a relevant outlook. In addition, the practical applicability in the field of smart farming will be investigated in more detail.

References

Alur, R. (1999). Timed Automata. In N. Halbwachs & D. Peled (Eds.), *Proceedings of the 11th international conference on computer aided verification (cav '99), july 6-10, 1999, trento, italy* (pp. 8–22). Springer Verlag.

Bano, D., Michael, J., Rumpe, B., Varga, S., & Weske, M. (2022). Process-aware digital twin cockpit synthesis from event logs. *Journal of Computer Languages*, *70*, 101121.

Behrmann, G., David, A., & Larsen, K. G. (2004). A Tutorial on Uppaal. In M. Bernardo & F. Corradini (Eds.), *Formal methods for the design of real-time systems: 4th international school on formal methods for the design of computer, communication, and software systems, sfm-rt 2004* (pp. 200–236). Springer–Verlag.

Bengtsson, J., & Yi, W. (2003). Timed Automata: Semantics, Algorithms and Tools. *Lectures on Concurrency and Petri Nets*, 87–124.

Broy, M., Damm, W., Henkler, S., Pohl, K., Vogelsang, A., & Weyer, T. (2012). Introduction to the SPES Modeling Framework. In K. Pohl, H. Hönninger, R. Achatz, & M. Broy (Eds.), *Model-Based Engineering of Embedded Systems* (pp. 31–49). Springer Berlin Heidelberg.

Buttazzo, G. C. (2011). *Hard real-time computing systems: Predictable scheduling algorithms and applications* (3rd ed) [OCLC: ocn741541202]. Springer.

Gausemeier, J., Rammig, F. J., & Schäfer, W. (Eds.). (2014). *Design methodology for intelligent technical systems: Develop intelligent technical systems of the future* (1st ed. 2014). Springer Berlin Heidelberg : Imprint: Springer.

Giese, H., Tichy, M., Burmester, S., Schäfer, W., & Flake, S. (2003). Towards the Compositional Verification of Real-Time UML Designs. *Proc. of the 9th european software engineering conference held jointly with 11th ACM SIGSOFT international symposium on foundations of software engineering (ESEC/FSE-11), Helsinki, Finland*, 38–47.

Henkler, S., & Eckardt, T. (2012). Component behavior synthesis for critical systems. In H. Giese, M. Huhn, J. Phillips, & B. Schätz (Eds.), *Dagstuhl-workshop MBEES: modellbasierte entwicklung eingebetteter systeme viii, schloss dagstuhl, germany, 2012, tagungsband modellbasierte entwicklung eingebetteter systeme* (pp. 113–122). fortiss GmbH, München.

Henzinger, T. A. (1992). Sooner is Safer than Later. *Information Processing Letters*, *43*(3), 135–141.

Henzinger, T. A., Nicollin, X., Sifakis, J., & Yovine, S. (1992). Symbolic Model Checking for Real-Time Systems. *Proceedings of the Seventh Annual Symposium on Logic in Computer Science (LICS)*, 394–406.

Hirsch, M., Henkler, S., & Giese, H. (2008). Modeling Collaborations with Dynamic Structural Adaptation in Mechatronic UML. *Proc. of the ICSE 2008 Workshop on Software Engineering for Adaptive and Self-Managing Systems (SEAMS"08), Leipzig, Germany*.

Lamport, L. (1977). Proving the Correctness of Multiprocess Programs. *IEEE Transactions on Software Engineering*, *SE-3*(2), 125–143.

Milner, R. (1989). *Communication and Concurrency*. Prentice-Hall, Inc.

Monteiro, J., Barata, J., Veloso, M., Veloso, L., & Nunes, J. (2018). Towards sustainable digital twins for vertical farming. *2018 Thirteenth International Conference on Digital Information Management (ICDIM)*, 234–239.

Sangiovanni-Vincentelli, A., Damm, W., & Passerone, R. (2012). Taming Dr. Frankenstein: Contract-Based Design for Cyber-Physical Systems*. *European Journal of Control*, *18*(3), 217–238.

Sreedevi, T. R., & Santosh Kumar, M. (2020). Digital twin in smart farming: A categorical literature review and exploring possibilities in hydroponics. *2020 Advanced Computing and Communication Technologies for High Performance Applications (ACCTHPA)*, 120–124.

Storey, N. R. (1996). *Safety critical computer systems*. Addison-Wesley Longman Publishing Co., Inc.

Verdouw, C., Tekinerdogan, B., Beulens, A., & Wolfert, S. (2021). Digital twins in smart farming. *Agricultural Systems*, *189*, 103046.

Yi, W., Pettersson, P., & Daniels, M. (1994). Automatic Verification of Real-time Communicating Systems by Constraint-solving. In D. Hogrefe & S. Leue (Eds.), *Proceedings of the 7th ifip wg6.1 international conference on formal description formal techniques, berne, switzerland, 1994* (pp. 243–258). Chapman & Hall.