Towards Scenario-Based Synthesis for Parametric Timed Automata

Holger Giese∗
Software Engineering Group, Department of computer science
University of Paderborn, Warburger Str. 100, 33098 Paderborn, Germany
E-mail: hg@uni-paderborn.de

Abstract

A number of approaches exists that permit to synthesize the operational state-based behavior of a set of components for a set of scenarios. In this paper we explore how to extend these approaches to also tackle the problem for scenarios with parametric timing constraints in form of upper and lower time bounds. The resulting time consistency problems are identified and ideas for an algorithmic handling are presented.

1. Introduction

Engineering large and complex systems often results in a number of scenarios developed to identify and describe the required behavior of the system components and its software. Several approaches [7, 8, 9] exploit the information provided by a given set of scenarios to synthesize the operational state-based behavior of the components.

In many domains timing information are an important aspect of the scenario descriptions. Thus we study in this paper how timing information such as worst-case execution times (WCET) or deadlines can be integrated when synthesizing operational models from scenarios. The operational target model in mind is a statechart variant with clocks named real-time statecharts. As they have a well defined real-time semantics based on timed automata [5] we will restrict our presentation to timed automata and omit more complex syntactical aspects such as hierarchical states.

In this paper a first approach towards the synthesis of parametric timed automata from scenarios with parametric timing constraints is presented. As the underlying synthesis problem is more complex than in the un-timed, non parameterized case, no scalable approach for large systems seems possible. Therefore, it is assumed that the scenarios describe the partial real-time behavior of roles within a pattern. Such patterns will contain only a limited number of components and thus synthesis remains feasible. The parameterized patterns are reusable units which can be employed later within the design when the specific coordination is required.

Some related work and the specific problems of timed scenarios are reviewed next. In Section 3 an example is used to illustrate the employed concepts and their formalization. Then for scenarios with explicit timing constraints the concepts for a synthesis algorithm are presented in Section 4. In Section 5 this algorithm is generalized to also cover parameterized constraints. The paper closes with a final conclusion and an outlook on future work.

2. Related Work

A number of notations for scenario description techniques with timing constraints exists such as message sequence charts with timing constraints [3], action diagrams (timing diagrams) [6] or UML sequence diagrams. Timing constraints are described in quite different ways (cf. [3]). Timers with reset and timeouts, delay intervals for events and activities, drawing rules, or timing markers in form of boolean expressions that constrain particular events or the whole diagram are possible options. In this paper we will only consider delay intervals as they permit an abstract and declarative descriptions of even parameterized timing constraints for scenarios. They permit to avoid the complexity which would result from arbitrary timing markers or any other general form of logic timing conditions.

If timing constraints are considered, causality has to be ensured (cf. [6]). Causality requires that for a single scenario a realization exists that can ensure correct operation without global knowledge or knowledge about the future. In the presented approach, we have to address this question even for the considered multiple overlapping scenarios.

For the scenario-based synthesis including timing constraints exists with [8] to the best knowledge of the author today only a very restricted proposal. The approach proposed in [8] synthesizes only a single automaton. The employed timed automata dialect [1] is further not sufficient to describe the intended operational behavior, because angelic non-determinism is assumed and no concept to specify progress conditions is supported.

When in contrast to [8] progress can be explicitly modelled, the models may include time stopping deadlocks. Such a deadlock occurs when a timed automaton reaches a configuration, where neither a time-less labelled transition nor a time-step can be taken. Conflicts within a given set of timing constraints are one possible reason for time stopping deadlocks. While structural conflicts can be detected without building the full state space of the system, the possibilities to consider reachability conflicts are in contrast rather restricted. The membership question (reachability) for timed automata is known to be PSPACE-complete [2] and thus efficient tool support for synthesis seems to be rather unrealistic. However, in an incremental scenario timed automata model checkers such as UPPAAL [4] can be employed to detect time stopping deadlocks of the synthesized models.

The situation for parametric timed automata is even more restricted. Emptiness, the question whether a fulfilling behavior for a syntactically correct parametric timed automata exists at all, has been proven to be undecidable for parametric timed automata with more than 2 parameters in [2]. Neverthe-
less, this strong limitation highlights that much more than the later presented concepts for detection of structural conflicts in the parametric case cannot be expected.

### 3. Timed Scenarios and Automata

As example we consider the simple watchdog pattern where a periodic operating controller component is connected to a watchdog. When the operation of the controller is as expected, it will periodically send messages to the watchdog to indicate that the controller is still in a sane state. If the watchdog does not receive such a message within a specified time frame, a reset of the controller will be initiated.

![Figure 1. Regular operation of the controller](image)

In Figure 1 a UML sequence diagram (extended by some explicit state annotations) is used to describe the expected regular operation of the controller role within the watchdog pattern. Each control cycle of the controller is determined by reading sensor input and writing actor control variables with a control period restricted by upper and lower time bounds $cp_a$ and $cp_l$. For the control algorithm the WCET $ca_a$ is estimated. Additionally, in each control cycle a sane message is sent to the connected watchdog via a channel. The channel introduces a buffering delay with WCET $bd_a$ and has to ensure message delivery from the controller to the watchdog with upper time bound $md_a$. The processing of the sane message by the watchdog has a WCET $wc_a$ and the minimal arrival time of sane messages is restricted using the lower time bound $at_l$.

![Figure 2. Watchdog timeout and reset](image)

The case that the watchdog does not receive the expected sane message within a time frame is described in Figure 2. A timeout occurs when the lower time bound $wt_l$ is reached. After a diagnosis step with WCET $wd_a$, the reset of the controller is initiated by the watchdog via a message reset. The channel will in any state (*) accept that reset message with priority and forward it to the controller. The controller will also accept the message in any state and executes some initialization code to restore a sane state with WCET $rc_a$ leading finally to the initial wait state. For the whole processing from the initial detection of the problem to the final successful reset of the controller we have the upper bound $rd_a$.

The basic problem of behavior synthesis from timed scenarios is to derive an appropriate set of timed automata $M_1, \ldots, M_i$ for each component for a given set of scenarios $Z_1, \ldots, Z_m$ with timing constraints. To first formalize scenarios we use two disjoint sets of events $E_a$ and $E_s$ to denote action and state observations, respectively. Together they build the set $E$ of observation events. Each event will be later mapped to either a state of the global state set $S$ or an action from the sets for communication actions $A$ or internal actions $I$. The events related to each component $M_i$ denoted by $E_i$ further build non disjoint subsets of $E$. A run $R$ is a tuple $(E_a, E_s, \triangleright)$ with $E_a \subseteq E_a, E_s \subseteq E_s$ sets of events and $\triangleright \subseteq E_a \times E_a \cup E_s \times E_a$ an acyclic graph ordering of event occurrences.

If we are not interested in the distinction between $E_a$ and $E_s$ we denote a run $(E_s, E_a, \triangleright)$ simply by $(E, \triangleright)$. We also define the action event ordering $\triangleright_a$ by $e \triangleright_a e'$ iff a $f \in E_a$ with $e \triangleright f \triangleright e'$ exists. The transitive closure $\triangleright^*$ of $\triangleright$ determines the partial ordering of event occurrences. For $R = (E, \triangleright)$ to be a syntactical correct run for component $M_i$ with related events $E_i$ we require that the sub-relation $\triangleright \cap E_i \times E_i$ is deterministic and thus describes the sequence of events related to component $M_i$.

To extended the concepts for scenarios w.r.t. time, a partial function $X : (E_a)^2 \rightarrow R_0 \times R_0^\infty$ with $R_0 = \{ r \in R | r \geq 0 \}$ and $R_0^\infty = R_0 \cup \{ \infty \}$ is further defined to be a valid timing constraint for $(E_a, E_s, \triangleright)$ iff it assigns only to such pairs of action events $(e, e') \in (E_a)^2$ with $e \triangleright^* e'$ and $e \neq e'$ a timing constraint $(a, b)$ with $a \in R_0, b \in R_0^\infty$, and $a \leq b$. We further restrict the constraint pair $(a, b)$ to non-trivial ones with $a \neq 0$ or $b \neq \infty$. In Figure 1 and 2 dashed lines and expressions like $\leq wd_a$ have been used to describe such timing constraints. A time mapping $\nu : E_a \rightarrow R_0$ assigns to each action event the single point of time when it is executed. For any $e \triangleright e'$ must hold $\nu(e) \leq \nu(e')$ to ensure that the execution time points respect the logical partial ordering. Note, that for a logical ordering described by a run $R$ infinite many time mappings are possible. Even when only the total ordering of events is considered, all possible interleavings of a run result in a number of different possible sequences. A timing constraint $X : (E_a)^2 \rightarrow R_0 \times R_0^\infty$ is respected by a time mapping $\nu : E_a \rightarrow R_0$ iff for all $(e, e') \in (E_a)^2$ with $X((e, e')) = (a, b)$ hold $a \leq (\nu(e') - \nu(e)) \leq b$.

**Definition 1** A timed run $((E_a, E_s, \triangleright), \nu)$ consists of an untimed run $R = (E_a, E_s, \triangleright)$ and a time mapping $\nu : E_a \rightarrow R_0$. A timed scenario $Z = (R, X)$ consists of a run $R$ and $X : (E_a)^2 \rightarrow R_0 \times R_0^\infty$ a valid timing constraint for $R$. For the set of all runs $R$ and the set of all scenarios $Z$ a timed run $(R, \nu) \in R$ is a realization for a timed scenario $Z = (R', X) \in Z$ iff $R = R'$ and $X$ is respected by the time mapping $\nu$. The timed run $(R, \nu)$ and the timed scenario $Z = (R', X)$ with $R = R'$ are otherwise in conflict.

For the formal foundation of the component behavior we use a simplification of the timed automata variant [5] with a dense time model built by a set of real value clocks $T$ with values in $R_0$. A clock binding $\mu : T \rightarrow R_0$ is used to denote the current value for a subset $T \subseteq T$ of all clocks. Addition-
ally, guards can be used to restrict the valid clock bindings for which the transition can be executed and clock updates can be used to reset clocks. For the global state set \( S \) we further define for a subset \( S \subseteq S \) a state invariant \( inv \) to determine the subset of valid time bindings for each state \( s \in S \). For \( \mu : T \rightarrow R_0 \) and \( \delta \in R_0 \) we define \( \mu \oplus \delta \) to be a clock binding with for any \( t \in T \) holds \( (\mu \oplus \delta)(t) = \mu(t) + \delta \). Additionally, the set \( R_0^\infty \) is used in guards \( g : T \rightarrow R_0 \times R_0^\infty \) to describe tests that for all \( t \in T \) with \( g(t) = (a, b) \) holds \( \mu(t) \in [a, b] \), where \( a = 0 \) and \( b = \infty \) respectively denote that no lower and upper bound is given. For a communication action \( a \in A \) we further use distinct send \((a!)\) and receive \((a?)\) labels. Thus we can define timed automata as follows:

**Definition 2** A timed automaton \( M = (S, T, inv, A, [\cdot], Q) \) is defined by a set of states \( S \subseteq S \), a set of real value clocks \( T \subseteq T \), a state invariant \( inv : S \rightarrow \psi([T \rightarrow R_0]) \), an alphabet \( A \subseteq A \cup A! \cup A? \) with for no \( a \in A \) holds \( a? \), \( a! \in A \), a step relation \([\cdot] \), and \( Q \subseteq S \) the non-empty set of initial states. The step relation \([\cdot] \) is \( S \times A \times \{ [T \rightarrow R_0 \times R_0^\infty] \times [\psi(T) \rightarrow R_0] \times \{ [\text{urgent}, \text{normal}] \} \times S \) further consists of a source state, a label, a time guard, a clock update, an urgent flag and a target state. We further define the set \( T_i^w \subseteq T \) of clocks written by clock updates and \( T_i^w \subseteq T \) to denote that ones read by guards or invariants.

![Figure 3. Timed automaton for the watchdog](image)

The presented formal concepts can be graphically represented as depicted in Figure 3. For sake of graphical representation we replace the verbose message names with their initial character and use numbering to distinguish messages with the same name but different source and target components (e.g., \( s_1 \) and \( s_2 \) will be used instead of same). Additionally, the sender and receiver sides are distinguished using \( s_2! \) respectively \( s_2? \). Urgent and non-urgent transitions are visualized by solid and dashed arrows, respectively. The timed automaton presented consists of only two states. The first one (ok) is initially entered with clock \( t_w \) set large enough to ensure that the lower inter arrival time bound \( at_l \) is fulfilled. The only transition of state ok is not enabled until a sane event can be received and the clock \( t_w \) has a value higher then the lower inter arrival time bound \( at_l \). If enabled it must be fired immediately (urgent) and the second state will be entered in an atomic step which also resets the clock \( t_w \) to zero. That the second state must be left before the WCET \( wc_u \) of the watchdog checking routine has elapsed, is denoted by the time invariant \( t_u \leq wc_u \). The related step leads back to the ok state.

In contrast to finite automata the state space of a timed automata is build by a state \( s \in S \) (often called location) and a clock binding \( \mu : T \rightarrow R_0 \). For \( (s, a, g, \mu_u, f, s') \in [\cdot] \) the regular time-less steps \( (s, \mu)(s', \mu') \) with \( \mu' \) such that for \( \mu : T \rightarrow R_0 \) and \( \mu_u : T' \rightarrow R_0 \) with \( T' \subseteq T \) for any \( t \in T' \) holds \( \mu'(t) = \mu_u(t) \) and otherwise \( \mu'(t) = \mu(t) \). The step can happen when \( \mu \) fulfills the guard \( g \) and \( (s', \mu') \) fulfills \( inv \). For transitions with \( f = \text{urgent} \) hold that they have to be executed immediately whenever enabled. The time step \( (s, \mu)(s, \mu \oplus \delta) \) can happen if \( (s, \mu \oplus \delta) \) fulfills \( inv \) and no urgent time-less transition is enabled for any \( (s, \mu \oplus \delta') \) with \( 0 \leq \delta' < \delta \). The configuration change describes that \( \delta \) time units have elapsed.

In order to address multiple timed automata \( M_1, \ldots, M_n \) we conveniently use \( S_i, T_i, inv_i, [\cdot]^i, A_i, \) and \( Q_i \) to denote the corresponding elements of \( M_i \). For any two \( i \neq j \) and the corresponding timed automata we require: that the alphabets \( A_i \) and \( A_j \) are disjoint, that the state sets \( S_i \) and \( S_j \) are disjoint, and that the written clock sets \( T_i^w \) and \( T_j^w \) are disjoint. Thus, we permit to share read clocks. This is used to emulate that deadline information can be embedded into the communication as for example planned in the current proposal for a distributed real-time Java [10]. Note, that if shared clocks are used their overlapping usage within timed automata can result in unexpected side-effects. A later synthesis has thus to ensure that the states of the scenarios are only mapped on each other, when this does not result in possible overlapping clock usage.

For given interactions \( A \), internal actions \( I \), and a finite number of timed automata \( M_1, \ldots, M_n \) we further define \( A_i \subseteq A \) the set of all \( a \in A \) for which \( i \neq j \) exists with \( a? \in A_i \) and \( a! \in A_j \) the local communication and name \( M_i \) the receiver and \( M_j \) the sender. The remaining actions \( A_e = A - A_i \) are external communication. The parallel composition of a set of timed automata is simply their synchronous execution w.r.t. the local communication \( (A_i) \).

We further restrict the set of considered timed automata to such ones where transitions with internal actions \( I \) and external communication \( (A_e) \) are always normal while all internal communication actions \( (A_i) \) are urgent and will happen immediately if the opposite side required for synchronization is also enabled.

For the execution of internal communication actions \( a \in A_i \) we assume that a scenario may imply additional restrictions (time guards) to ensure proper processing as long as timing analysis excludes conflicts. For external communication actions and internal actions \( i \in I \) in contrast, we can only assume the worst case delay resp. WCET. Thus the upper bound can actually occur and has to be taken into account.

Using the above definitions and assumptions we can consider how to synthesize timed automata from scenarios with timing constraints.

### 4. Synthesis for Time Constraints

The underlying fundamental question of scenario-based synthesis is whether a set of timed automata realize a given scenario. Informally, a timed run is included in the behavior of a set of timed automata when for the timed sequence of events a corresponding sequence of configurations and steps can be constructed for the set of automata. To address this question we proceed as follows: We first require the additional formalization of a state mapping between the scenarios and a system of timed automata. Then, we can define what it means for a system of timed automata and a given mapping
that a run is included (Definition 3). We can then formally define the timed synthesis problem (Definition 4). Then the construction of the constraint graph resulting from a specific mapping of scenarios to a set of timed automata is described. That a valid constraint graph excludes conflicts is then shown in Lemma 1 and 2. Finally, how the additional time guards required for the synthesis are computed using the constraint graph is described in Algorithm 1.

A mapping map = (map⁰, map¹, map²) will be used to encode how the different events are related to states and actions resp. transitions of the timed automata.¹ For the mappings hold that map⁰ assigns to each element of 𝒦 a corresponding element of 𝒢, map¹ assigns to each element of 𝒸 a corresponding element of 𝒜 ∪ 𝒢, and map² assigns to each element of 𝒢 a single or two corresponding transitions ([{1, ..., n}]) and ({[1, ..., n]}))². The mapping must respect the system decomposition such that states, actions, and transitions of 𝑀ℓ can only be assigned to events of 𝒦.

For a run 𝑅 = (E, ▽) an event ∈ 𝒦 is further defined to be initial iff no ′ ∈ 𝒦 with ′ ∈ 𝒦 exists. We denote the set of initial observations of 𝑅 by initial(𝑅). We define a run 𝑅 = (E, ▽) to be complete iff for the initial observations hold initial(𝑅) ⊆ 𝒦 and for all i we have |initial(𝑅) ∩ 𝒦|= 1. We further define the pre-condition of an event Pre((E, ▽), ) = {′ ∈ 𝒦 | ▽′ ∈ E}. For a run 𝑅 = (E, ▽) the execution of an event ∈ initial(𝑅) does result for 𝑃′ = 𝑃(𝑅, ) ∈ ′ the run 𝑃′ = 𝑃′(𝑅, ) ∈ ′ denoted by Exec(𝑅, ).

Definition 3 Timed automata 𝑀ℓ= |...| 𝑀𝑛 ∈ a state ((s1, ..., s𝑛), μ) and a global clock value t include the complete run 𝑅 = ((E, , ▽), ) for mapping map if (i) The set of initial events corresponds to the state: {s1, ..., s𝑛} = map⁰(′| ∈ initial(𝑅)). (ii) For ∈ 𝒦, ∈ 𝒦 with ∈ initial(𝑅) and ∈ 𝒦 ♦ ∈ 𝒦 ♦ with minimal ♦(′) and ♦ = ♦(′) − t, a = map¹(′), a ∈ 𝒜, s1 = map²(′), and s′ = map²(′) hold ((s1, ..., s𝑛), μ) ((s1, ..., s𝑛), μ ⊕ ♦ | ♦(′)) and ((s1, ..., s𝑛), μ ⊕ ♦ | ♦(′)) via map²(′). Additionally, the complete run Exec(𝑅, ) for the global clock value t + ♦ must be included in the resulting state ((s′1, ..., s′n), μ′).

The synthesis problem also taking timing constraints into account can thus be defined as follows:

Definition 4 The timed synthesis problem for sets 𝒜, 𝒢, 𝒦 etc. and a given set of complete scenarios 𝑍₁, ..., 𝑍ₘ is to derive timed automata 𝑀₁, ..., 𝑀ₙ for each component such that runs 𝑅₁, ..., 𝑅ₘ with 𝑅ᵣ realizes 𝑍ᵣ exists and no run 𝑅ᵣ of 𝑀₁ |...| 𝑀ₙ exists which is in conflict with any 𝑍ᵣ.

A problem which can result from a set of timing constraints is that no solution exists. By exploiting the structural information of the scenarios we can detect some of these conflicts. We restrict our attention here to such ones which will also lead to a conflict in the synthesized model to exclude false negatives. To detect all conflicts, a reachability analysis of the model has to be done which is in general only decidable for the non parameterized case.

To exclude structural conflicts we next describe how to build the related constraint graph. For a set of timed automata {𝑀ᵢ}, a set of timed scenarios {𝑍ᵢ}, a mapping (map⁰, map¹, map²), an additional timed scenario 𝑍 = ((E, , ▽), ) and a specific constraint (, ′) ∈ 𝑋 with non trivial lower and upper time bounds (a, b) = ((, ′), ) we define the constraint graph GC = (E, ▽, ) with 𝑋 ⊆ 𝐸, ▽ ∈ E², and : E² → ℜ₀ as follows:

(1) 𝐸 = { ∈ 𝐸| ▽ ⊔ 𝑓 ∧ ▽ ⊔ ′} denotes all action events which are located on a path between and ′.

(2) We start with 𝑐 initialized such that it returns zero for all (, ′) ∈ 𝐸² and ▽ = ∅. Then, for each (, ′) ∈ 𝐸², Z′ ∈ {𝑍ᵢ} with Z′ = ((E′, E′, ▽), X′) and each specific constraint (, ′) ∈ 𝑋 with (, ′) = X′((, ′)) we check the following: If a path = e₁...ek...ek+...ek+1...ek+m in 𝑍 with e₁ = ☑ and ek and e’ = ′ and a path = ′...e’ exists with for all 1 ≤ i ≤ l hold map² ek+i = map² e’i we add (′, ′) to ▽ and update the value of c(′, ′).

Informally, we are looking for all lower and upper bound constraints that restrict a part of a path in the underlying timed automata model leading from to ′. The upper bound is added to the constraint graph if a single non-urgent transition is detected. Otherwise we use the lower bound only. The reason is that we will later add appropriate guards which will enforce that the urgent transitions will be executed early enough to still ensure that the required upper bound will not be violated. Using the constraint graph we can use the following fact to exclude conflicts due to upper bounds.

Lemma 1 For any constraint (a, b) = X((, ′)) the upper time bound b cannot be missed when for all path = e₁...ek from to ′ in the related scenario and the related constraint graph c(e₁, e₂) + ... + c(eₖ₋₁, eₖ) ≤ b holds.

Proof: (sketch) For non-urgent transitions returns the maximal upper bound and thus the statement obviously holds. If also events related to internal communication are involved, each path can in principle be blocked. However, when running backwards through the constraint graph we can for every possibly blocked path (due to internal communication) simply choose the non-blocked alternative. Thus only all non-blocked paths have to be considered to ensure that the upper bound cannot be missed.

The following observation also exploited in [8] permits to ignore the lower bound case of the structural consistency problem altogether:

¹This mapping will be computed in a first un-timed synthesis step.

²The steps which might behave erratic are either internal steps (的步伐) or external communications (𝐴ᵣ). All constraints with at least one local transition (and thus with a path longer than two) will themselves ensure that all contained erratic elements will not hurt their upper bound and thus can be omitted.
Lemma 2 For a timed automaton a lower time bound \( a \) for 
\((e, e')\) with \((a, b) = X((e, e'))\) cannot be missed when adding \( a \leq t \leq b \) to the guard of the transition related to \( e' \).

Proof: (sketch) The guard guarantees the lower time bound, 
because we can always wait for \( a \leq t \) to hold. \( \square \)

Using both lemmata we can exclude conflicting runs. Note, 
that the existence of fulfilling runs as required by Definition 
4 is, however, not guaranteed.

A number of different proposals for a synthesis algorithm 
exists. The general scheme is to process a list of given scenar-
ios and successively adding their behavior. To synthesize and 
check the set of timed automata for a set of timed scenarios 
we follow this idea and proceed as follows to add a scenario:

1. In a first step we can process the un-timed elements of 
the model using existing proposals such as [7, 9] with minor 
adjustments to compute the required mapping \( \text{map} \) for action 
events and state events.

2. For the synthesis algorithm we better avoid the rather 
cumbrous procedure described in the definition of the con-
straint graph which checks for the timing constraints by pro-
cessing all scenario structures. Instead, a global constraint 
data-structure \( C \) storing all constraints using the state and 
transition structure of the set of timed automata results in a 
more efficient realization. We use such a global constraint 
data-structure and a function \( \text{updateGCD} \) to efficiently up-
date it and a related function \( \text{constraintGraph} \) to effi-
ciently compute the required constraint graph. The required 
processing for \( \{M_i\} = \{M_1, \ldots, M_n\} \) the set of timed au-
tomata and \( C \) the global constraints data-structure contain-
ning the already processed scenarios is then as follows:

Algorithm 1

```
addTimeScenario(((M_i), C), map, ((E_u, E_d, D), X))
begin
  forall (e, e') ∈ X do // forall constraints
    (a, b) := X((e, e'));
    t_o := newClock();
    (M_i) := addClockPlusReset{\{M_i\}, map'(e), t_o};
    (M_i) := addGuard{(M_i), map'(e'), a \leq t \leq b};
    (E_u, E_d, D) := constraintGraph(e', ((E_u, E_d, D), X), C);
    D := \{(e', 0)|P := (e'); // D = max delay map
while (P \neq E) do // add guards
  F := \{f ∈ E | P \subseteq f' \subseteq P'; f' > f\}.
  forall f ∈ F do
    d := \max\{d+(f', f')|f' ∈ P ∧ (f', d) ∈ D ∧ f' > f\};
    if (d > b) then
      ERROR("conflict", (E, D, c), f);
    fi
    (M_i) := addGuard{(M_i), map'(f), t_o \leq b - d};
    D := D \cup \{(f, d)\}; // add delay
  done
  P := P \cup F;
  done
done
return \{(M_i), updateGCD{C, ((E_u, E_d, D), X)}\};
end
```

The presented algorithm iterates over the constraint set of 
the scenario and proceeds per constraint as follows: It first adds 
a shared clock to the set of automata and ensures that this 
clock is reset to zero for the transition related to the initial ac-
tion event \( e \) (addClockPlusReset). Additionally, for the 
final transition \( (map'(e')) \) the required time guard is added 
(addGuard). Then the maximum delay of all in Lemma 1 
considered possible paths between two action events is com-
puted and the inequality is tested. Additionally, for each
reached transition the guard is extended (addGuard) such 
that the transition will only be executed when still enough 
time is left for the remaining paths to \( e' \). While the number 
of paths in the scenario may be exponential, to compute the 
maximal delay also taking constraints of earlier added scenar-
ios into account requires roughly \( O(|E| + |D|) \) in the forAll 
loop by simply propagating the maximal delay via the con-
straint graph.

From Lemma 1 and 2 follows that the parallel composition 
\( M_1 \parallel \ldots \parallel M_n \) of the synthesized timed automata 
computed by a correct synthesis algorithm and extended by Algorithm 1 
for the scenarios \( Z_1, \ldots, Z_m \) cannot contain a run which is in 
conflict with any \( Z_i \).

For the in Figure 4 presented synthesized timed automata, 
the following constant upper and lower bounds \( m_d = 3 \) ms, 
\( b_d = 2 \) ms, \( e_p = 200 \) ms, \( e_p = 180 \) ms, \( c_u = 140 \) ms, 
\( w_c = 15 \) ms, \( a_d = 80 \) ms, \( w_d = 8 \) ms, \( w_d = 270 \) ms, 
\( r_d = 30 \) ms, and \( r_e = 15 \) ms have been used. We further must 
split the internal action with upper bound \( c_u \) into two 
separate upper bounds \( c_a_1 = 70 \) ms and \( c_a_2 = 70 \) ms to 
reflect the fact that the send operation is an intermediate step. 
Both upper bounds together ensure the required upper bound 
for the whole activity. Following the thread of control and 
message flow for the case of a reset initiated by the watchdog 
we can see how the described backward propagation works. 
Starting in state error of the timed automata of the watchdog 
the clock \( t_{wd} \) is successively tested for 8, 13, 15, and 30 in 
the different timed automata for the watchdog, channel, and 
controller to ensure that intermediate delays cannot result in 
an execution where the required deadline can be missed.

However, also some limitations of the proposal show up.
When a scenario terminates, no information about the implied 
clock binding for the resulting final states is given. While for 
the watchdog an appropriate solution can be found that does 
not restrict the further behavior, for the controller and channel 
their upper time bounds restrict the time until their normal 
processing has to be started.

It is to be noted, that scenario descriptions for reusable 
patterns will usually contain only a limited number of explicit 
timing constraints, because for the most elements of the sys-
tem timing information is not available that early in the 
development process. Therefore, parametric time restrictions
have to be used which can be handled in a similar fashion as presented next.

5. Synthesis for Parametric Time Constraints

To formulate and analyze the parametric timing constraints we first introduce the concept of linear inequalities: A system of linear inequalities for \( n \) variables \( x_1, \ldots, x_n \) over \( \mathbb{R} \) is defined by \( m \) inequalities with coefficients \( a_{ij} \in \mathbb{R} \) of the form \( a_{i1}x_1 + \cdots + a_{in}x_n \leq b_i \). A set of linear inequalities is feasible iff a solution \( v \in \mathbb{R}^n \) exists. Feasibility of linear inequalities can be solved by variants of the simplex algorithm as well as the polynomial interior point methods.

The concepts defined for the timed case with constant upper and lower time can be straightforward extended to the parametric case using inequalities. Instead of constants \((c_a, c_b) \in \mathbb{R}_0 \times \mathbb{R}_0^n\) we instead employ variables \((x_a, x_b) \in \mathbb{R}^2\) with domain \( \mathbb{R}_0 \). To abstract from the usage of either variables \((x_a)\) as well as constants \((c_a)\) we use in the following simply terms \((v_a)\). When at least one element of \((v_a, v_b)\) is a variable the simple inequality \( v_a \leq v_b \) has to be added. Analogously to the non parameterized case we can then synthesize and check the parametric timed automata for a set of timed scenarios. In step (1) only un-timed elements of the model have been considered and thus the same procedure as in the non parameterized case can be employed. To compute in step (2) the maximum delay of all in Lemma 1 considered possible paths between two action events of a restriction \((e, e')\) in addTImedScenario simply the maximum is propagated. Due to the employed variables, however, a direct computation of the terms of all required inequalities would result in an exponential worst-case complexity. Therefore, we add fresh variables \( x_f \) for a node \( f \) if multiple edges with distinct terms \( v_i \) exist and add the inequalities \( v_i + c((f, f')) \leq x_f \) for each \( v_i \). We then simply propagate \((f, x_f)\) via the delay set \( D_f \). If only a single term \( v \) has to be propagated we propagate \((f, v')\) with \( v' = v + c((f, f')) \). Thus we can still do the computation in \( O(|E_c|+|P_r|)\) steps per constraint by propagating the resulting terms in the linear case and using a new variable otherwise. For each newly added variable and when the source event \( e \) is reached, an inequality \( v_i \leq x_f \) respectively \( v_i \leq v_b \) has been added. Thus the maximal delay for each path from \( e \) to \( e' \) is bound by the resulting set of inequalities to be less than \( v_b \).

The user can incrementally add available information in form of additional explicit or parametric timing constraints. As long as the system of linear inequalities derived by the presented algorithm is feasible, there exist possible timing parameters that exclude structural conflicts.

In the controller example, for instance, the inequalities will ensure that the sum of the upper bound for the watchdog diagnosis step \( w_d_a \), the channel internal upper bound for the buffering delay \( b_d_a \), and upper bound for the controller reset \( r_e_a \) is always less than the upper bound of the reset deadline \( r_d_a \). If thus, for example, the buffering delay \( b_d_a \) is defined to be larger than \( r_e_a \) (e.g., due to a concrete WCET and required reset times) the obvious conflict results as expected in the infeasibility of the system of linear inequalities.

6. Conclusions and Future Work

The presented algorithms for the case of explicit and parametric timing constraints permit to add available timing information in an incremental manner. It ensures that timing conflicts (at least structural ones) are reliably detected when adding another scenario or when adjusting the timing constraints of one that has already been considered. However, the problem of reachability conflicts remains and is probably best tackled when instantiating a pattern such that the parameters are replaced by concrete values. Then a model checker for timed automata such as UPPAAL \([4]\) can be used to detect them.

The current proposal is only a conceptual study and it remains to be proven that reasonable large sets of scenarios for a pattern can also be handled. Therefore, the prototypical implementation of the proposed algorithm within the Real-Time Fujaba-framework (http://www.fujaba.de) is planned as a next step. Additionally, a tighter integration between the state mapping questions and timing constraints seems fruitful.

Acknowledgements

I wish to thank Sven Burmester, Ekkart Kindler, and Matthias Tichy for comments on earlier versions of this position paper.

References


