Compositional Random Testing using Extended Symbolic Transition Systems

Christian Schwarzl[1], Bernhard K. Aichernig[2], and Franz Wotawa[2]

[1] Virtual Vehicle,
Inffeldgasse 21a, 8010 Graz, Austria,
christian.schwarzl@v2c2.at
[2] Institute for Software Technology,
Graz University of Technology, 8010 Graz, Austria,
aichernig@ist.tugraz.at, wotawa@ist.tugraz.at

Abstract. The fast growth in complexity of embedded and software enabled systems requires for automated testing strategies to achieve a high system quality. This raise of complexity is often caused by the distribution of functionality over multiple control units and their connection via a network. We define an extended symbolic transition system (ESTS) and their compositional semantics to reflect these new requirements imposed on the test generation methods. The introduced ESTS incorporates timed behavior by transition execution times and delay transitions. Their timeout can be defined either by a constant value or an attribute valuation. Moreover we introduce a communication scheme used to specify the compositional behavior and define a conformance relation based on alternating simulation. Furthermore we use the conformance relation as the basis for a simple random test generation technique to verify the applicability of the presented approach. This formal framework builds the foundation of our UML test case generator.

Keywords: symbolic transition system, concurrent reactive behavior, test case generation

1 Introduction

Since testing is an important task to ensure a certain system quality, the used techniques and approaches in this field strongly advanced in recent years. Nevertheless testing still remains a laborious task and is – due to the high degree of manual interaction – error prone. The steadily increasing complexity of embedded systems requires a high degree of test automation to be able to execute and analyze the large amount of needed test cases.

A further increase in automation can be achieved by the generation of test cases from formal specifications, which has gained a lot of attention in research in recent years and becomes more and more popular in the industry. However, the currently available industrial- and scientific-tools based on unified modeling language (UML) state machines or symbolic transition systems (STSs) [6] are

limited to a single model. This situation does not meet the requirements of modern embedded systems, which often consists of communicating components.

For this reason we have implemented a test case generation algorithm, which works on the basis of an extended symbolic transition system (ESTS) composition presented in this work. This prototype supports a systematic and randomized test generation approach, which detailed description is beyond the focus of this paper.

Our contribution in this work is the extension of the STS by delay- and completion-transitions, timing groups, transition priorities and their execution duration. In addition we provide a precise semantics and formally define the model composition. Furthermore we show the applicability of the presented symbolic framework by a randomized test generation approach and the conformance relation to the system under test (SUT).

The remainder of the paper is structured as follows: Section 2 defines the structure of an ESTS and Section 3 precisely describes the compositional behavior. In Section 4 a conformance relation based on alternation simulation is provided and the applicability of the presented approach is demonstrated in Section 5. Section 6 presents an overview of the related work and Section 7 concludes the paper and gives a short outlook.

2 Extended Symbolic Transition System

In this section we define the structure of an ESTS and its semantics with respect to its contained states and transitions. Based on this structure we explain the creation of traces through the ESTS caused by external interactions.

Definition 1 (Extended Symbolic Transition System). *An ESTS is a tuple* $\langle \mathcal{S}, \Lambda, \mathcal{A}, \mathcal{P}, \mathcal{T}, \mathcal{G}, s_0, \iota_0 \rangle$*, where* $\mathcal{S}$ *is a set of states,* Λ *are the signals,* $\mathcal{A}$ *are the attributes,* $\mathcal{P}$ *are the signal parameters,* $\mathcal{T}$ *is the transition relation and* $\mathcal{G}$ *is a set of timing groups. Moreover* s_0 *is the initial state and* ι_0 *is the initial attribute valuation.* [3] □

We define the set of signals $\Lambda = \Lambda_i \cup \Lambda_o$ as the union of input Λ_i and output Λ_o signals. The set $\Lambda_* = \Lambda \cup \{\tau, \gamma, \delta\}$ is the complete set of all defined signals, whereas the constants $\tau, \gamma, \delta \notin \Lambda$ represent the special unobservable-, completion- and delay-signal types, respectively. The attributes $\mathcal{A}$ are the variables of an ESTS and the parameters $\mathcal{P}$ are variables attached to input- or output-signals $\lambda \in \Lambda$. Further it holds that $\mathcal{A} \cap \mathcal{P} = \emptyset$ and we use $V = \mathcal{A} \cup \mathcal{P}$.

The transition relation of an ESTS is defined as $\mathcal{T} \subseteq \mathcal{S} \times \Lambda_* \times \mathfrak{F}(V) \times \mathfrak{T}(V) \times \mathfrak{P} \times \mathcal{S}$, where $\mathfrak{F}(V)$ is a first order logic formula without quantifiers, $\mathfrak{T}(V)$ are attribute value assignment statements given as mathematical terms over the variables V and $\mathfrak{P}$ are the priorities.

[3] In [6] a different naming convention is used, where states are locations, transitions are switches, attributes are location variables and parameters are interaction variables.

We write a transition $t \in \mathcal{T}$ as $s \xrightarrow{\lambda,\varphi,\rho,p_t} s'$, where $s \in \mathcal{S}$ is its source state, $s' \in \mathcal{S}$ is the destination state, $\lambda \in \Lambda_*$ defines its type, $\varphi \in \mathfrak{F}(V)$ is the guard, the action $\rho = \langle \rho_1, \rho_2, .., \rho_n \rangle$ is an ordered list of assignment terms $\rho_j \in \mathfrak{T}(V)$ with list index j and $p_t \in \mathfrak{P}$ is its priority, where $p_t \in \mathbb{N}_0$. A transition $t \in \mathcal{T}$ has a traversal probability $\alpha_t \in \mathbb{R}$ and an execution duration $d_t \in \mathbb{N}_0$ in addition. We omit the presentation of the priority p_t in the remainder for simplicity if the context allows it.

Definition 2 (Timing Group). *A timing group g is a tupel $\langle c, S_\delta, T_\delta, T_r \rangle$, where c is its clock, $S_\delta \subseteq \mathcal{S}$ are the contained states, $T_\delta \subseteq \mathcal{T}$ are the delay transitions and $T_r \subseteq \mathcal{T}$ are the clock reset transitions.* □

A timing group $g \in \mathcal{G}$ specifies a set of states S_δ where each of these states has an outgoing delay transition $t_\delta \in T_\delta$ with the same timeout $n_\delta \in \mathbb{N}_0$. The states in the timing group share a clock c, which is used to trigger the traversal of one of these outgoing delay transitions. The timeout of a delay transition can either be defined by a constant- or by an attribute-value like $n_\delta = 100$ or $n_\delta = x$ if $x \in \mathcal{A}$. Our delay transitions should not be confused with the similar delay transitions in timed automata semantics. In timed automata semantics, delay transitions serve to express the possible waiting times in a state and hence are reflexive transitions increasing time only. Our delay transitions increase time and change the state.

We require that every state $s \in S_\delta$ has an outgoing delay transition $t_\delta \in T_\delta$ to the same destination state $s' \in \mathcal{S}$. This ensures that one of the delay transitions is traversed after the defined amount of time – specified by the timeout of the delay transitions – within the timing group has elapsed. The timing group clock is set to zero if one of the clock reset transitions $t_r \in T_r$ is traversed.

We use $\vartheta = \iota \cup \varsigma$ as variable valuation containing the values of attributes ι and the current signal parameters ς. Accordingly we denote the update of a variable valuation ϑ' according to an action ρ as $\vartheta \mapsto \vartheta'(\rho)$. Given a valuation ϑ and a guard φ we write $\vartheta \models \varphi$ if the valuation satisfies the guard φ, which is a first order logical formula.

Example 1. Figure 1 shows an illustrative example of two communicating ESTS, where we use ? to mark input- and ! for output-signals, the keyword `delay(x)` to indicate delay transitions, the parameter x to denote the delay time and show only the guard of the completion transition γ. Blocking states as given in Definition 4 are shown with two border lines and states belonging to the same timing group are filled gray. Since each of these ESTSs contains only one timing group their presentation is unambiguous. Transition guards are shown within squared brackets and actions follow a slash. □

Definition 3 (Configuration). *A configuration q is a tuple $\langle s, \iota \rangle$, where $s \in \mathcal{S}$ is an explicit state and ι specifies the values of all attributes in $\mathcal{A}$.* □

A configuration fully defines the current state of an ESTS and accordingly we define the initial state $q_0 = \langle s_0, \iota_0 \rangle$, where s_0 is the initial state and ι_0 the

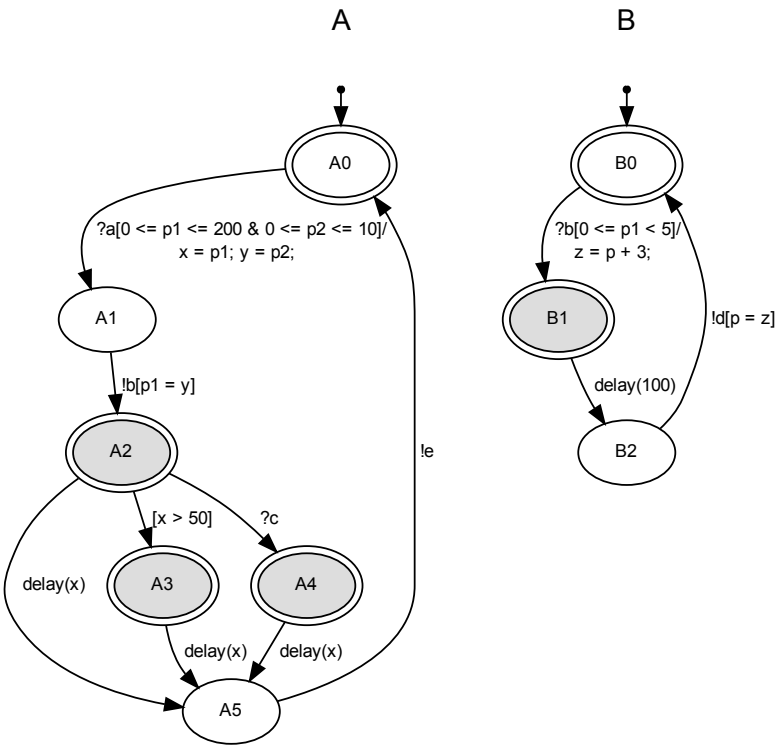

Fig. 1. Two communicating ESTSs A and B

initial valuation. In the remainder we use $\mathcal{Q}$ to indicate the set of all possible configurations.

For the following definitions we use $t \in \mathcal{T}$, $\lambda \in \Lambda$, $\lambda_* \in \Lambda_*$ and $g \in \mathcal{G}$. The function $\texttt{src}(t)$ returns the transition source state, $\texttt{dest}(t)$ its destination state, $\texttt{signal}(t)$ the signal of a transition, $\texttt{arity}(\lambda) \in \mathbb{N}_0$ the number of signal parameters and $\texttt{dur}(t)$ the execution duration d_t.

In addition we use $\texttt{out}(s, \lambda_*) = \{t \in \mathcal{T} \mid \texttt{signal}(t) = \lambda_* \wedge \texttt{source}(t) = s\}$ returning all outgoing transitions of s having signal λ_*, $\texttt{delay}(t) = n$ if $\texttt{signal}(t) = \delta$ or $n = 0$ otherwise, providing the delay time of a transition and $\texttt{delay}_{min}(t) = min\left(\bigcup \texttt{delay}(t_\delta) \mid t_\delta \in \texttt{out}(\texttt{src}(t), \delta)\right)$, which is is the minimum delay of currently active delay transition.

The function $\texttt{prio}(t)$ returns the transition priority p_t and $\texttt{prio}_{max}(s, \lambda) = max\left(\bigcup\{\texttt{prio}(t) \mid t \in \texttt{out}(s, \lambda)\}\right)$ calculates the maximum priority of all outgoing transitions from state s having signal λ.

$\texttt{clk}(t) = \bigcup\{\texttt{c}(g) \mid \texttt{src}(t), \texttt{dest}(t) \in \texttt{states}(g) \wedge t \notin \texttt{r}(g)\}$ returns the set of all timing groups to which the given transition belongs.

The function $\texttt{reset}(t) = \bigcup\{\texttt{c}(g) \mid t \in \texttt{r}(g)\}$ defines the clocks which are reset by a traversal of t, where $\texttt{c}(g)$ returns the clock c, $\texttt{r}(g)$ the clock reset transitions T_r and $\texttt{states}(g)$ the contained states S_δ of the given timing group g.

Example 2. Let the ESTS A shown in Figure 1 be initialized with the configuration $q_0 = \langle A0, \{x = 0, y = 0\}\rangle$. After the traversal of the transition $A0 \xrightarrow{?a[0\leq p1\leq 200 \& 0\leq p2\leq 10]/x=p1;y=p2;} A1$ with the parameters $p1 = 70$ and $p2 = 6$ the ESTS A has the configuration $q' = \langle A1, \{x = 70, y = 6\}\rangle$ as defined by the transition action. □

2.1 Semantics

This section describes the behavior of every allowed transition type, which is defined by the prerequisites of the transition traversal and the performed state and attribute value changes.

We define the semantics of an ESTS in the Rules (1) to (6), where we require $\vartheta \models \varphi$ if not stated differently, we use $Q' \subseteq \mathcal{Q}$ as the set of configurations after a transition traversal and $\mapsto$ denotes an assignment mapping. The shown semantics is similar to the one presented by Frantzen et. al, but we only describe the evaluation of the post state to highlight the extensions.

Rule *Empty* (1) states that the state and the valuation does not change if the empty signal ϵ is executed, where $\top$ indicates that the guard is always satisfied and $\texttt{id}$ is the identity function.

$$\frac{s \xrightarrow{\epsilon,\top,\texttt{id}} s}{Q' \mapsto \{\langle s, \vartheta\rangle\}} \tag{1}$$

In *Input* (2) the semantics of a signal reception $\lambda_i \in \Lambda_i$ is defined, where a signal reception can only be executed if it is sent before an active timeout. Rule *Timed* (3) shows that a delay transition with the shortest delay is executed as soon as the defined amount of time has elapsed. After the traversal the current state and the attribute valuation are updated and the clocks of the timing groups where $t \in T_r$ are set to zero.

$$\frac{s \xrightarrow{\lambda_i,\varphi,\rho,p_t} s' \wedge p_t = \texttt{prio}_{max}(s, \lambda_i) \wedge \forall c \in \texttt{clk}(t) \mid c < \texttt{delay}_{min}(t)}{Q' \mapsto \{\langle s', \vartheta'(\rho)\rangle\}, \forall c \in \texttt{clk}(t) \mid c \mapsto c + \texttt{dur}(t), \forall c \in \texttt{reset}(t) \mid c \mapsto 0} \tag{2}$$

$$\frac{s \xrightarrow{\delta,\varphi,\rho,p_t} s' \wedge p_t = \texttt{prio}_{max}(s, \delta) \wedge \texttt{c}(t) \geq \texttt{delay}(t) \wedge \texttt{delay}(t) = \texttt{delay}_{min}(t)}{Q' \mapsto \{\langle s', \vartheta'(\rho)\rangle\}, \forall c \in \texttt{clk}(t) \mid c \mapsto c + \texttt{delay}(t) + \texttt{dur}(t), \forall c \in \texttt{reset}(t) \mid c \mapsto 0} \tag{3}$$

The rules *Output* (4) and *Completion* (5) have the same behavior in terms of updating configurations and the handling of the clock updates as in (2), but are executed as soon their guard is satisfied. They differ in the IO behavior, because in (4) a signal is sent and (5) only allows for a deterministic configuration update.

$$\frac{s \xrightarrow{\lambda_o,\varphi,\rho,p_t} s' \wedge p_t = \texttt{prio}_{max}(s,\lambda_o)}{Q' \mapsto \{\langle s', \vartheta'(\rho)\rangle\}, \forall c \in \texttt{clk}(t) \mid c \mapsto c + \texttt{dur}(t), \forall c \in \texttt{reset}(t) \mid c \mapsto 0} \tag{4}$$

$$\frac{s \xrightarrow{\gamma,\varphi,\rho,p_t} s' \wedge p_t = \texttt{prio}_{max}(s,\gamma)}{Q' \mapsto \{\langle s', \vartheta'(\rho)\rangle\}, \forall c \in \texttt{clk}(t) \mid c \mapsto c + \texttt{dur}(t), \forall c \in \texttt{reset}(t) \mid c \mapsto 0} \tag{5}$$

Unobservable (6) defines the semantics of a non-deterministic configuration update. The traversal of an τ transition is not observable and the resulting symbolic states in Q' are its source and destination state.

$$\frac{s \xrightarrow{\tau,\varphi,\rho,p_t} s' \wedge p_t = \texttt{prio}_{max}(s,\tau)}{Q' \mapsto \{\langle s, \vartheta\rangle, \langle s', \vartheta'(\rho)\rangle\}, \forall c \in \texttt{clk}(t) \mid c \mapsto c + \texttt{dur}(t), \forall c \in \texttt{reset}(t) \mid c \mapsto 0} \tag{6}$$

2.2 Simulation

In this section the execution of an ESTS is explained by the creation of execution traces caused by signal receptions or delays. Such execution traces are always embedded between two blocking states described in Definition 4.

Definition 4 (Blocking State). *A blocking state $\tilde{s}$ is a state $s \in \mathcal{S}$ for which it holds that* $\exists \lambda \in \lambda_i \cup \delta \mid |\texttt{out}(s,\lambda)| \neq 0 \vee \texttt{out}(s,\lambda_*) = \emptyset$. □

This means a blocking state is a state having at least one outgoing transition of type λ_i or δ or no outgoing transition $\lambda_* \in \Lambda_*$ at all. Accordingly we denote a blocking configuration as $\tilde{q} = \langle \tilde{s}, \iota \rangle$. Note that a blocking state does not limit the occurrence of outgoing τ, γ or $\lambda_o \in \Lambda_o$ transitions, which makes a mixed state possible. Since we allow outgoing output transitions, the state is not a *quiescent* state as defined in [6] or [11].

Based on Definition 4 we can define an execution trace as shown in Definition 5, which connects two blocking states and must not be interrupted. It is a sequence of transitions and consists of a triggering η_t and completion η_c part, where η_t consists only of transitions with signals $\Lambda_t = \Lambda_i \cup \delta$ and $\Lambda_c = \Lambda_o \cup \gamma \cup \tau$.

Definition 5 (Execution Trace). *An execution trace $\eta = t_1, t_2, \ldots, t_n$ is a sequence of transitions $t_1, \ldots, t_n$, where $\eta_t = t_1$, $\eta_c = t_2, \ldots, t_n$ and* $\forall t_i \in \eta \setminus t_1 \mid \texttt{dest}(t_i) = \texttt{source}(t_{i+1})$. □

The length of an execution trace is denoted as $|\eta|$ and it holds that $|\eta| \geq 1$, where $|\eta_t| = 1$ and $|\eta_c| \geq 0$. Due to the allowed non-deterministic behavior of an ESTS a signal reception or a time lapse can cause multiple execution

traces leading to the resulting list $\mathcal{E}(\tilde{q}, \lambda)$. Its recursive generation is defined by $\mathcal{E}'(\tilde{q}, \lambda) = \bigcup\{e(\eta) \mid \eta \in \mathcal{E}(\tilde{q}, \lambda)\}$, where $\lambda \in \Lambda_c$ is the triggering input, $\tilde{q} \in \mathcal{Q}$ is the current configuration and $\mathcal{E}'(\tilde{q}, \lambda)$ is initialized with $t \in \texttt{out}(\tilde{q}, \lambda) \mid \vartheta \models \varphi$. The function $e(\eta)$ is defined in Equation (7), where $\tilde{q}'$ is the destination state of the last contained transition in η and $\circ$ is the concatenation of traces.

$$e(\eta)' = \begin{cases} \bigcup\limits_{t_c}\{e(\eta \circ t_c) \mid t_c \in \texttt{out}(\tilde{q}', \Lambda_c)\} & \text{if } \texttt{out}(\tilde{q}', \Lambda_t) = \emptyset \wedge \texttt{out}(\tilde{q}', \Lambda_c) \neq \emptyset \\ \eta \cup \bigcup\limits_{t_c}\{e(\eta \circ t_c) \mid t_c \in \texttt{out}(\tilde{q}', \Lambda_c)\} & \text{if } \texttt{out}(\tilde{q}', \Lambda_t) \neq \emptyset \wedge \texttt{out}(\tilde{q}', \Lambda_c) \neq \emptyset \\ \eta & \text{otherwise} \end{cases} \tag{7}$$

The recursive generation of the completion steps in (7) creates an infinite number of traces if a loop of completion transitions exists, which actions do not falsify one of its guards.

Example 3. Let ESTS A shown in Figure 1 again be initialized with $q_0 = \langle A0, \iota_0 \rangle$, where $\iota_0 = \{x = 0, y = 0\}$. Then we can build the list of initial execution traces $\mathcal{E}(\tilde{q_0}, \lambda) = A0 \xrightarrow{?a[0 \leq p1 \leq 200 \& 0 \leq p2 \leq 10]/x=p1;y=p2;} A1$, because we use $\lambda = a$, $\vartheta = \iota_0 \cup \varsigma$ and $\varsigma = \{p1 = 30, p2 = 9\}$, which satisfies the transition guard. If we would use $\varsigma = \{p1 = 30, p2 = 15\}$ instead, then $\mathcal{E}(\tilde{q}, \lambda) = \emptyset$ because $p2 > 10$. The recursive update of this list leads to the final trace list $\mathcal{E}(\tilde{q_0}, \lambda) = A0 \xrightarrow{?a[0 \leq p1 \leq 200 \& 0 \leq p2 \leq 10]/x=p1;y=p2;} A1 \xrightarrow{!b[p1=y]} A2 \xrightarrow{[x>50]} A3$, where the first transition is the triggering- η_t and the last two transitions are the completion- η_c part. The last transition has to be added, because $\vartheta = \{x = 30, y = 9\} \models \varphi$ of the completion transition, which is executed immediately after its guard is satisfied. □

3 Composition

In this section the model composition based on the signal communication between the involved ESTSs is explained. Furthermore we clearly define the observations and interactions, which can be made by the environment.

3.1 Model Communication

In this work we use a deterministic communication scheme using a global queue Q to pass signals between ESTSs. Since we required that an execution trace must not be interrupted, the system behavior can be described by an concatenation of such traces. This concept is similar to the approach presented in the language Creol [3], where only one thread is active at a time.

The reception of a signal λ or the lapse of time in the state $\tilde{q}$ leads to a list of execution traces $\eta \in \mathcal{E}'(\tilde{q}, \lambda)$. The needed execution time or the sent signals by a trace η can cause reactions in other ESTSs or the environment. Therefore

a list of system execution traces $\mathcal{E}_M(\tilde{q}, \lambda)$ has to be created, where $M \subseteq \mathcal{M}$ is the set of all involved ESTSs. These traces contain the initial trace η and its concatenated reaction traces of the other ESTSs.

Since not every signal needs to be sent or be observable by the environment – e.g. communication within one component – we split the signals into two categories. The first category contains signals observed or created by the environment $\bar{\Lambda} = \bar{\Lambda}_i \cup \bar{\Lambda}_o \mid \bar{\Lambda}_i \subseteq \Lambda_i \wedge \bar{\Lambda}_o \subseteq \Lambda_o$ to which we refer as *external communication* in the remainder. The second category is the *internal communication*, which contains the signals $\hat{\Lambda} = \hat{\Lambda}_o \cup \hat{\Lambda}_i \mid \hat{\Lambda}_o = \Lambda_o \setminus \bar{\Lambda}_o \wedge \hat{\Lambda}_i = \Lambda_i \setminus \bar{\Lambda}_i$ sent and received by one of the ESTSs. Note that signals part of the external communication *must* be and signals of the internal communication *might* be created or received by the environment.

The trace output signals used for the communication are defined in Definition 6 and are independent of the observability by the environment.

Definition 6 (Observables). *Let $obs(\eta) = \langle\langle \lambda_o, \overline{d}_1 \rangle, \ldots, \langle \lambda_o, \overline{d}_N \rangle\rangle$ be the output signals of a trace $\eta \in \mathcal{E}'(\tilde{q}, \lambda)$, where its result is a list of tuples, N is the number of output signals on the trace, $\lambda_o \in \Lambda_o$ are the signals and $\overline{d}$ is the time period in which the signal has to be sent. We define the observable-$\overline{obs}(\eta) = obs(\eta) \mid \lambda_o \in \bar{\Lambda}_o$ and unobservable $\widehat{obs}(\eta) = obs(\eta) \mid \lambda_o \in \hat{\Lambda}_o$ output signals of η.* □

The required state updates in the other ESTSs are performed using the signals in $obs(\eta)$, which are passed via the global message queue Q. The execution uses the same algorithm as described above and leads to the execution trace $\eta_m \in \mathcal{E}_m(\tilde{q}, \lambda)$ in the ESTS $m \in M$. We call a trace $\eta_M \in \mathcal{E}_M(\tilde{q}, \lambda)$ containing the initial execution trace η and all according reactions η_m *system execution trace* in the remainder.

Definition 7 (System Execution Trace). *A system execution trace $\eta_M = \eta_{m_1} \circ \cdots \circ \eta_{m_N}$ is the catenation of execution traces η_{m_i} of an ESTS $m \in M$, where $i = 1..N$, $N = |M|$ is the number of entries in M and $M \subseteq \mathcal{M}$ is the set of involved ESTS.* □

$$rt(\eta) = \begin{cases} \eta & \text{if } obs(\eta) = \emptyset \\ \bigcup\limits_{\eta'} \{\eta \circ \eta' \mid \eta' \in rt\left(\eta_m \in \mathcal{E}_m(\tilde{q}_m, \lambda) \mid \lambda \in obs(\eta)\right)\} & \text{otherwise} \end{cases} \tag{8}$$

The creation of system execution traces is defined recursively by the reception trace function $rt(\eta)$ shown in Equation (8), where $\tilde{q}_m$ is the current configuration of $m \in M$. It shows that a system trace is a recursive concatenation of all execution traces of other ESTSs caused by the output signals on the initial path. Given the initial traces $\eta \in \mathcal{E}(\tilde{q}, \lambda)$, we can build a list of all possible execution traces $\mathcal{E}_M(\tilde{q}, \lambda) = \bigcup rt(\eta) \mid \eta \in \mathcal{E}(\tilde{q}, \lambda)$

This algorithm ensures that all signals stored in the queue are processed before the next execution step can begin. However, each execution trace can also

produce output signals, which are also enqueued as described above. This allows for the creation of infinite loops between the involved ESTS, where a signal reception causes an output signal received in another ESTS, which reception causing the initial signal sending responsible for the initial stimuli.

Since the reception of signals has a higher priority than the traversal of delay transitions, their influence was neglected during the finding of the system execution traces. Due to the fact that the traversal of a transition t needs the time $\texttt{dur}(t)$ to be completed, the execution time is calculated in the function $\texttt{time}(\eta)$ given in (9).

$$\texttt{time}(\eta) = \sum_{i=1}^{N} \texttt{dur}(t_i) + \texttt{delay}(t_i) - \texttt{time}(\eta_g(t_i)) \tag{9}$$

In (9) η_g is the connected sub-trace of η consisting of the transitions contained in the timing group g to which the transition t_i belongs. Since an execution trace can also contain transitions not belonging to the ESTS m, which is the owner of the timing group g, these transitions still have to be included. A formal definition of the sub-trace creation is given in (10), where i is the index of t_i in η. It uses (11) to extract the trace from η according to the given indices and (12) to find the start index of the trace based on (13) returning the transitions of the ESTS to which the transition t_i belongs.

$$\eta_g(t_i) = \texttt{sub}(\eta, i, k) \mid k = \texttt{idx}_e(t_i, T_m), T_m = \texttt{gtrace}(\eta, t_i) \tag{10}$$

Using Definition 5 a sub-trace of η is given by (11), which consists of the transitions in η lying in the range $[i, j]$, which is defined by the given indices.

$$\texttt{sub}(\eta, i, j) = \langle t_i, .., t_j \mid t_i, t_j \in \eta \wedge 1 \leq i, j \leq |\eta| \wedge j \geq i \rangle \tag{11}$$

In (12) the minimum index k of the of the given transition t in T_m is calculated, which references to the first transition of the trace stored in T_m.

$$\texttt{idx}_e(t, T_m) = k \in \mathbb{N} \mid (\exists t_k \in T_m \mid k = min_{idx}(T_m)) \tag{12}$$

The function $\texttt{gtrace}(\eta, t_i)$, as defined in (13), returns all transitions and their indices in η satisfying the following criteria, where g is the timing group belonging to t_i. The first term of the constraint $t_j \in \eta$ requires that the transition belongs to η and the second term $\texttt{src}(t_j), \texttt{dest}(t_j) \in \texttt{states}(g)$ that the transitions are contained in the same timing group as t_i. Term three $\texttt{dest}(t_j) = \texttt{src}(t_{j+1})$ ensures that the transitions represent a trace without any structural holes. The last term $\exists t_j = t_i$ requires that the given transition t_i is contained in that connected trace to prevent an ambiguous result if t_i is traversed multiple times in trace η.

$$\begin{aligned} \texttt{gtrace}(\eta, t_i) = \bigcup \{ t_j \mid & t_j \in \eta \wedge \texttt{src}(t_j), \texttt{dest}(t_j) \in \texttt{states}(g) \\ & \wedge \texttt{dest}(t_j) = \texttt{src}(t_{j+1}) \wedge \exists t_j = t_i \} \end{aligned} \tag{13}$$

The elapsed time $\texttt{time}(\eta_M)$ is used to trigger active delay transitions after the processing of the enqueued output signals has finished. This is done by finding the transition with the smallest time overdue $\delta_{due} = \texttt{delay}(t) - \texttt{time}(\eta_M)$. If such a transition exists it is executed using the same algorithm as described above. The execution can again cause the sending of new output signals, which are processed before the next delay transition is taken into account. This algorithm again allows for the modeling of an infinite loop, if two traces exist with $\texttt{time}(\eta_1) \geq \texttt{delay}(t_2)$ and $\texttt{time}(\eta_2) \geq \texttt{delay}(t_1)$ and which lead to their own source state, where η_1 and η_2 are the execution traces to the transitions t_1 and t_2, respectively. The execution traces gained from the processing of the delayed transitions are then concatenated to η being the final result.

In this approach we defined that the treatment of a signal reception has a higher priority than the traversal of an delayed transition. These rules allow that an active delay transition, whereas enough time has elapsed to trigger the traversal, is not traversed in favor to the transition receiving a signal from Q, even if the signal was enqueued after the timeout of a delayed transition.

4 Conformance

In this section the correctness of an implementation under test (IUT) with respect to a specification using alternating simulation [1] is explained. For simplicity, we only discuss the conformance of deterministic ESTS here. In the non-deterministic case the two ESTSs need to be determinized beforehand, similar to [12]. Generally it is required that the IUT can follow all inputs generated from and only produces outputs allowed by the specification. To provide a precise understanding we introduce the function $\texttt{moves}(\tilde{q}, \Lambda)$ shown in Equation (14) first, where $M \subseteq \mathcal{M}$ is a set of ESTSs, $\tilde{q}_M = \bigcup \tilde{q}_m \mid m \in M$ and $\tilde{q}_m$ is the blocking configuration of an ESTS $m \in M$. This function returns the union of all outgoing transitions of all $m \in M$ at state $\tilde{q}_m$, which guard is satisfied and signals are contained in the given set Λ.

$$\texttt{moves}(\tilde{q}_M, \Lambda) = \bigcup \{t \mid \texttt{src}(t) = \tilde{q}_M \wedge \tilde{q}_M \models \varphi_t \wedge \texttt{signal}(t) \in \Lambda\} \tag{14}$$

The meaning of alternating simulation as defined in [12] is formalized in Equation (15) and (16), where $\tilde{q}_1 \in \mathcal{Q}_1$ and $\tilde{q}_2 \in \mathcal{Q}_2$ are the sets of configurations of the IUT and the specification, respectively. Accordingly $\lambda_1 = \texttt{signal}(t_1)$ and $\lambda_2 = \texttt{signal}(t_2)$ are the signals of these transitions and q_1' and q_2' are the destination configurations.

$$\forall t_2 \in \texttt{moves}(\tilde{q}_2, \bar{\Lambda}_i \cup \delta) \mid (\exists t_1 \in \texttt{moves}(\tilde{q}_1, \bar{\Lambda}_i \cup \delta) \mid \lambda_2 = \lambda_1 \wedge \tilde{q}_2' = \tilde{q}_1') \tag{15}$$

Equation (15) states that all input or delay transition traversable in the specification in state $\tilde{q}_2 \in \mathcal{Q}_2$, which has a certain attribute valuation must also be executable on the IUT.

$$\forall t_1 \in \texttt{moves}(\tilde{q}_1, \bar{\Lambda}_o) \mid (\exists t_2 \in \texttt{moves}(\tilde{q}_2, \bar{\Lambda}_o) \mid \lambda_1 = \lambda_2 \wedge \tilde{q}_1' = \tilde{q}_2') \tag{16}$$

The inverse is true for outputs as shown in Equation (16), where it is required that every output produced by the IUT must be allowed by the specification. If both equations hold, then the basic I/O behavior of the implementation is correct with respect to the specification.

Since Equation (15) and (16) do not provide any information on the time behavior, we require in addition that the obtained output of the IUT fulfills the the timing constraints given in the ESTSs. Therefore we require that Equation (17) holds, where $\eta' \in \mathcal{E}_2(\tilde{q}_2, \texttt{signal}(t_2))$ and $\eta \in \mathcal{E}_1(\tilde{q}_1, \texttt{signal}(t_2))$ are the execution traces created for a given input on the IUT and specification respectively.

$$\forall \langle \lambda_o, d \rangle_j \in \overline{obs}(\eta) \mid (\exists \eta' \mid \langle \lambda'_o, \bar{d} \rangle_j \in \overline{obs}(\eta') \wedge \lambda_o = \lambda'_o \wedge min(\bar{d}) \leq d \leq max(\bar{d})) \tag{17}$$

Equation (17) requires that for every observable output λ_o at time d part of the trace η produced by the IUT an according transition $t'_o \in \eta'$ exists, which contains the same outputs λ'_o within the time range $\bar{d}$. In (17) $j = 1..|\eta|$ is the index of the output occurrence in η. The time range $\bar{d}$ is given in Equation (18), which is the sum of the execution time elapsed up to the transition at position $k = \texttt{idx}(t'_o, \eta')$ and includes a transition time jitter ε_k.

$$\bar{d} = \texttt{time}(\texttt{sub}(\eta', 1, k)) \pm \varepsilon_k \tag{18}$$

Since we have now defined the required outputs including the occurrence time of the IUT after an input was provided by the specification, we can now check the correctness of the IUT with respect to a given specification. Alternating simulation has the advantage in comparison to *ioco* that the conformance check is computational less intense and provides the same expressive power in the deterministic case [12].

5 Application

We show the applicability of the presented approach on a simple random test case generation example based on the ESTSs shown in Figure 1. In this example the external communication consists of the signals $\bar{\Lambda}_i = \{a, c\}$ and $\bar{\Lambda}_o = \{d\}$ and the internal communication is given by $\hat{\Lambda}_i = \{b\}$ and $\hat{\Lambda}_o = \{b\}$.

In this random approach named *random walk*, we explicitly trigger the traversal of outgoing transitions from the current state $\tilde{q}$. This is done by the generation of feasible data for the transitions $t \in \texttt{moves}(\tilde{q}, \bar{\Lambda}_i)$ and the lapse of time for delay transitions. The input generation is done separately for each transition t by a constraint solver e.g. provided by GNU Prolog as in our case, which tries to find solutions satisfying the transition guards.

If multiple transitions are possible during this phase, meaning they leave the current state and their guard can be satisfied, we normalize their probabilities α and perform a random selection. Since we also want to generate sequences, which vary in their temporal behavior, the random selection of an explicit *wait*

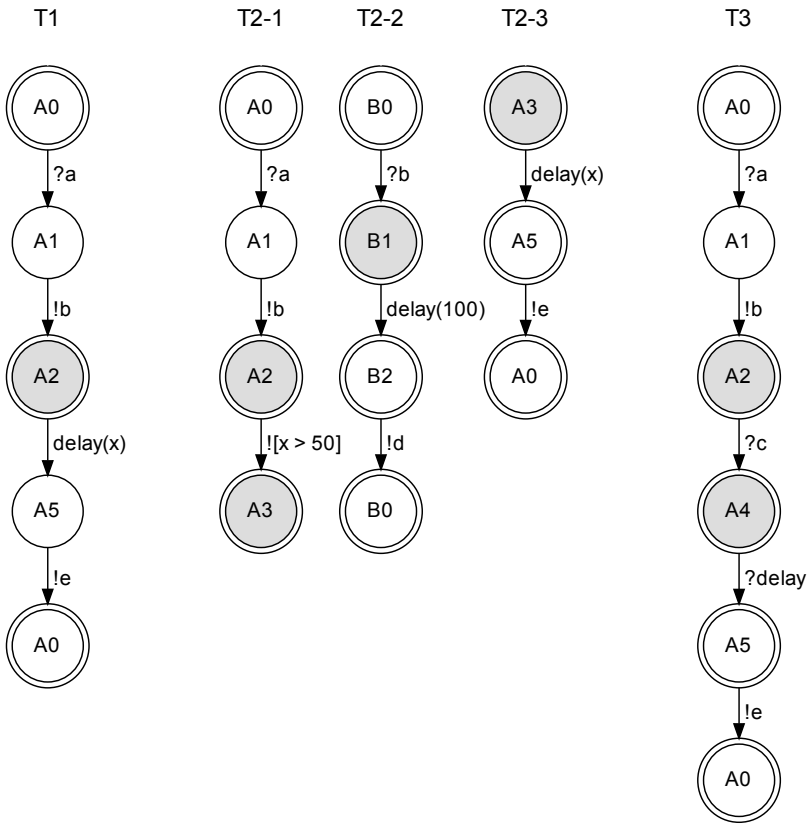

Fig. 2. Three system execution traces $T1, T2$ and $T3$

can also be chosen. In such a case the used wait time t_w has to be $0 \leq t_w \leq \delta_{min}$, where δ_{min} is the smallest timeout of all active delay transitions.

In the case an input action has been selected it is sent to and executed on the according ESTS. Wait actions in contrast are executed on the whole system, because the smallest active delay can belong to any of the involved ESTSs. After the input or wait action was performed the system execution traces $\mathcal{E}_M(\tilde{q}, \lambda)$ are generated.

Figure 2 shows three traces, which can be obtained if the inputs are applied as given in Table 1 and Table 2. The inputs in Table 2 lead to the same trace $T3$, but in the second case no additional wait is necessary, because the execution

	T1	**T2**
Inputs	$?a(p1 \leq 50, p2 \geq 5)$ $wait(p1)$	$?a(p1 > 100 + 4 * t_d, p2 < 5)$ $wait(100)$ $wait(p1 - (100 + 4 * t_d))$
$\overline{obs}(\eta)$	$\{\langle e, p1 + 4 * t_d \rangle\}$	$\{\langle d(z + 3), 100 + 6 * t_d \rangle, \langle e, p1 + 4 * t_d \rangle\}$
$\widehat{obs}(\eta)$	$\{\langle b, 2 * t_d \rangle\}$	$\{\langle b, 2 * t_d \rangle\}$

Table 1. Inputs and outputs for traces $T1$ and $T2$

	T3-1	**T3-2**
Inputs	$?a(t_d < p1 \leq 50, p2 \geq 5)$ $?c$ $wait(p1 - (100 + 2 * t_d))$	$?a(p1 \leq t_d, p2 \geq 5)$ $?c$
$\overline{obs}(\eta)$	$\{\langle e, p1 + 4 * t_d \rangle\}$	$\{\langle e, p1 + 4 * t_d \rangle\}$
$\widehat{obs}(\eta)$	$\{\langle b, 2 * t_d \rangle\}$	$\{\langle b, 2 * t_d \rangle\}$

Table 2. Two possible inputs and outputs for trace $T3$

time is longer than the required delay. For these examples we assumed that ever transition has the same execution duration $t_d = 10$.

Both tables also show the observable output generated by each trace, which can be used during the execution of the test case on the SUT. Since it also includes the latest point in time of the real signal reception, it is possible to check given timing constraints.

The random walk can be used for on-the-fly and offline test case generation. During on-the-fly testing the SUT is executed in parallel to the model and the input and outputs can be processed immediately. The advantage of this approach is that the current state is always known, which limits the state space especially in the presence of non-determinism. For offline test case generation all possible traces have to be stored and extended with every step during the random walk. Since non-determinism is allowed, the possible execution traces can be seen as a tree. This requires that the random walk has to continue in one step from every active leaf of this tree, to generate feasible test cases. Depending on the model these trees can become quite big due to the high number of possibilities.

6 Related Work

Several approaches based on symbolic transition systems have been studied in recent years. STG [4] is a symbolic extension of the test tool TGV and allows the generation of test cases with respect to a given test purpose. The presented framework extends the approach described in [6], by timed behavior, completion transitions and model composition. The approach in [6] is implemented in the STSimulator, which provides a framework for on-the-fly random testing and is used in the Jambition Project [5] to automatically derive test cases for web

applications. It uses **sioco** as conformance relation, which is the symbolic variant of **ioco** based on labeled transition systems (LTS).

Although the approaches described above were used successfully in various applications, they do not incorporate time as part of the specification. For this reason several extensions were introduced to lift the well understood approaches to timed models. This lead in the case of an LTS to its timed version and the according implementation relations like **tioco** and **rtioco**. A detailed discussion is given in [8], where a survey about the similarities and differences between these approaches and their variants is provided. However, these techniques still rely on an enumerative treatment of data limiting the scalability in data intense applications.

Also model checkers based on timed automata (TA) like UPPAAL [7,2] were used for behavior specification and test case generation. The timing constraints in a timed automata are given as time invariants on states and clock guards on transitions. UPPAAL also allows the interaction of data and time, meaning that attribute values can be used in the timing constraints. Their approach still relies on an explicit modeling of data and therefore faces the same scalability problems as methods based on an LTS. For the generation of test cases UPPAAL requires a deterministic specification, which limits the range of applicable use cases.

A symbolic variant based on timed automata is defined in [14], where the symbolic timed automata (STA) is introduced. It is a combination of an STS with the timing handling of TA and also allows the usage of attribute values as bounds for timing constraints. On the basis of the STA the testing conformance relation **stioco** being an symbolic extension of **tioco** is described. Although an STA allows similar semantics, it does not include a formal description of a composition and neglects unobservable events at the moment.

Spec Explorer [13], which can also use Spec# as specification language, uses alternating simulation to define the conformance between the IUT and the model. It was recently extended to work with UML sequence diagrams used for testing and program slicing. It also supports model composition in a similar way and allows the generation of test sequences based on a model composition. In contrast to the presented work no timed behavior can be modeled, which is which is one of the key features of the presented approach. Since Spec Explorer does no full symbolic state space exploration it allows a wider range of supported data types in contrast to this work, where we are limited to integer and boolean values.

7 Conclusion

We presented in this work an extended symbolic transition system based on the STS defined in [6]. Our approach extends this framework by the incorporation of delay- and completion-transitions for which we also provide a formal semantics. On top of the ESTS we defined a communication scheme, which uniquely defines the compositional behavior. In contrast to [6] we use alternating simulation as testing relation instead of **ioco**, for which we used the distinction between internal- and external-communication. This distinction allows for a clear sepa-

ration between observable or controllable signals by the environment and those used internally.

We used this symbolic framework for a sample application allowing a random test case generation, which can be performed on-the-fly and offline. The incorporation of delay transitions and transition execution times allows for timing checks of the SUT like the verification of trace files containing time stamps.

The presented ESTS in this work contains similar elements as defined in UML state machines and therefore allows for a straight forward model transformation. For this reason it can be used as a formalization of the UML state machine semantics, which is required for test case generation. This is the first time we presented the formal framework on which basis we have implemented our test case generation prototype from UML state machines. Parts of the tools chain and its application on industrial use cases have been described in [9] and [10].

Future work includes the investigation of other communication schemata and an extension of the transition attributes to allow uncertainties in the timed behavior like $t_d = 100 \pm 5$. This would allow for checks ensuring that a certain signal did not arrive before a given point in time, which is required for modeling real time networks.

Acknowledgment

The authors wish to thank the reviewers, the "COMET K2 Forschungsförderungs-Programm" of the Austrian Federal Ministry for Transport, Innovation and Technology (BMVIT), the Austrian Federal Ministry of Economics and Labour (BMWA), Österreichische Forschungsförderungsgesellschaft mbH (FFG), Das Land Steiermark and Steirische Wirtschaftsförderung (SFG) for their financial support.

References

1. Alur, R., Henzinger, T.A., Kupferman, O., Vardi, M.Y.: Alternating refinement relations. In: Proceedings of the 9th International Conference on Concurrency Theory. pp. 163–178. CONCUR '98, Springer-Verlag, London, UK (1998), http://portal.acm.org/citation.cfm?id=646733.759544
2. Behrmann, G., David, A., Larsen, K.G.: A tutorial on UPPAAL. In: Bernardo, M., Corradini, F. (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. No. 3185 in LNCS, Springer (September 2004)
3. de Boer, F., Clarke, D., Johnsen, E.: A complete guide to the future. In: De Nicola, R. (ed.) Programming Languages and Systems, Lecture Notes in Computer Science, vol. 4421, pp. 316–330. Springer Berlin / Heidelberg (2007), http://dx.doi.org/10.1007/978-3-540-71316-6_22
4. Clarke, D., Jéron, T., Rusu, V., Zinovieva, E.: STG: A symbolic test generation tool. In: Lecture Notes in Computer Science. pp. 151–173. Springer (2002)

5. Frantzen, L., Las Nieves Huerta, M., Kiss, Z.G., Wallet, T.: On-the-fly model-based testing of web services with Jambition. In: Bruni, R., Wolf, K. (eds.) Web Services and Formal Methods, pp. 143–157. Springer-Verlag, Berlin, Heidelberg (2009), http://dx.doi.org/10.1007/978-3-642-01364-5_9
6. Frantzen, L., Tretmans, J., Willemse, T.A.C.: Test generation based on symbolic specifications. In: FATES 2004, number 3395 in LNCS. pp. 1–15. Springer-Verlag (2005)
7. Hessel, A., Larsen, K.G., Mikucionis, M., Nielsen, B., Pettersson, P., Skou, A.: Testing real-time systems using UPPAAL. In: Hierons, R.M., Bowen, J.P., Harman, M. (eds.) Formal methods and testing, pp. 77–117. Springer-Verlag, Berlin, Heidelberg (2008), http://portal.acm.org/citation.cfm?id=1806209.1806212
8. Schmaltz, J., Tretmans, J.: On conformance testing for timed systems. In: Cassez, F., Jard, C. (eds.) Formal Modeling and Analysis of Timed Systems, Lecture Notes in Computer Science, vol. 5215, pp. 250–264. Springer Berlin / Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-85778-5_18
9. Schwarzl, C., Peischl, B.: Static- and dynamic consistency analysis of UML state chart models. In: Model Driven Engineering Languages and Systems. Lecture Notes in Computer Science, vol. 6394, pp. 151–165. Springer Berlin / Heidelberg (2010)
10. Schwarzl, C., Peischl, B.: Test sequence generation from communicating UML state charts: An industrial application of symbolic transition systems. In: Proceedings of the 2010 10th International Conference on Quality Software. pp. 122–131. QSIC '10, IEEE Computer Society, Washington, DC, USA (2010), http://dx.doi.org/10.1109/QSIC.2010.22
11. Tretmans, J.: Test generation with inputs, outputs, and quiescence. In: Lecture Notes in Computer Science. pp. 127–146. Springer (1996)
12. Veanes, M., Bjørner, N.: Alternating simulation and IOCO. In: Proceedings of the 22nd IFIP WG 6.1 international conference on Testing software and systems. pp. 47–62. ICTSS'10, Springer-Verlag, Berlin, Heidelberg (2010), http://portal.acm.org/citation.cfm?id=1928028.1928033
13. Veanes, M., Campbell, C., Grieskamp, W., Schulte, W., Tillmann, N., Nachmanson, L.: Model-based testing of object-oriented reactive systems with Spec Explorer. In: Hierons, R., Bowen, J., Harman, M. (eds.) Formal Methods and Testing, Lecture Notes in Computer Science, vol. 4949, pp. 39–76. Springer Berlin / Heidelberg (2008), http://dx.doi.org/10.1007/978-3-540-78917-8_2, 10.1007/978-3-540-78917-8_2
14. Von Styp, S., Bohnenkamp, H., Schmaltz, J.: A conformance testing relation for symbolic timed automata. In: Proceedings of the 8th international conference on Formal modeling and analysis of timed systems. pp. 243–255. FORMATS'10, Springer-Verlag, Berlin, Heidelberg (2010), http://portal.acm.org/citation.cfm?id=1885174.1885193