SPECIFICATION-BASED TESTING OF REAL-TIME EMBEDDED SYSTEMS*

Manuel Núñez and Ismael Rodríguez
Departamento Sistemas Informáticos y Programación, Facultad de Informática
Universidad Complutense de Madrid, 28040 Madrid, Spain
{mn,isrodrig}@sip.ucm.es

Abstract In this paper we present a formal framework to test real-time embedded systems where temporal requirements play a relevant role. First, we introduce our formal model to specify this kind of systems. It is a modification of the classical EFSM formalism where actions will be attached with a temporal constraint. This constraint expresses the maximal and/or minimal time this action should take during its execution in the current configuration. Then, we use this formalism to construct models that represent the set of temporal requirements in a system. In turn, they are automatically obtained from the temporal requirements defined for each component in the system. Tests for checking these requirements are defined and a formal mechanism to apply them to real systems is described.

1. Introduction

Embedded systems that are constructed nowadays have reached a high level of complexity. They may consist of a huge amount of components that interact with each other and may have been designed by different teams or companies. In this context, guaranteeing the reliability of each component and/or the integration of all of them is a great challenge. In particular, the integration of components yields an explosion of relevant configurations to be considered and checked, and selecting the most critical ones by hand is not feasible. The systematic testing of systems has been a topic of research and application of formal methods for years. Formal methods provide systematic testing mechanisms where systems are faced to some selected foreseen situations. Then, the correctness of the system is assessed by comparing its behavior with the one provided by a set of requirements or a specification. Due to the high cost of

*Research partially supported by the Spanish MCYT project TIC2003-07848-C02-01, the Junta de Castilla-La Mancha project PAC-03-001, and the Marie Curie project MRTN-CT-2003-505121/TAROT.

testing in any development project, the use of systematic and formal methods for testing has attracted the attention of several manufacturers.

In the beginning, specification languages proposed to create models of systems by considering only its functional characteristics. That is, models expressed constraints as *"after an action A is produced, an action B should/must be produced as well."* Thus, formal testing was constrained to check functional properties (see e.g. [Lee and Yannakakis, 1996] for a good survey on testing finite state machines). Besides, there were also some proposals to specifically deal with embedded and/or component-based systems (see e.g. [Petrenko et al., 1996; Lima and Cavalli, 1997]). Nevertheless, this kind of properties turned out to be insufficient for representing other critical features of systems. To cope with this problem, new models were developed to deal with additional characteristics such as the time consumed by actions, the probability of taking an action, the resources consumed to perform an action, or the probability that an action takes a given time to be performed. However, the inclusion of temporal issues in specification languages did not have a great impact on the development of new testing methodologies.

In this paper we present a formalism based on classical EFSMs (*extended finite state machines*) whose aim is to facilitate the specification of embedded systems where temporal aspects play a relevant role. In addition, we will present the basic concepts of a testing framework for these systems. The application of tests to check temporal issues and the obtention of diagnosis results concerning their correctness is defined. Our language will provide us with the possibility of separately describing the behavior of each component of a system. An automatic transformation mechanism that allows to create a single model from the models of some interacting components will be provided as well. This mechanism will not consider any possible combination of configurations of components. On the contrary, only *reachable* situations will be considered, which will diminish the usual state explosion problem.

Our formalism allows to express constraints about the time consumed by each action. Variables will affect the behavior of components and systems as well as the time consumed by actions. In particular, variables will have the capability to enable/disable actions. Every action in the model will be equipped with a temporal constraint expressing the (required) behavior for that action. For example, temporal constraints on actions could require than an action a takes *less* than 2 seconds, next action b takes *more* than 3 seconds and *less* than 5 seconds, and, finally, action c takes *exactly* 7 seconds. Let us suppose that the sequence of actions a, b, c is a possible communication sequence inside the system and that this sequence is observed from the outside of the system as the observable action d. If an external observer tries to check the temporal correctness of a system according to that requirement, then she should observe that the action d never takes less than 10 seconds nor more than

14 seconds. Let us note that the accomplishment of that requirement does not imply that each individual action in the sequence a, b, c accomplishes its own specific requirement. That is, if the temporal constraint for d is fulfilled, an external observer can only claim that the temporal correctness of actions a, b, c is *possible*. Let us note that if a *black box* testing approach is considered (see e.g. [Myers, 1979]) then it is irrelevant whether the temporal constraints on a, b, and c are fulfilled as long as the temporal behavior of the only observable action (d) is correct. Let us suppose now that there is no upper bound for the time consumed by b. Then, the only temporal constraint for d would require that it does not take less than 10 seconds. Our testing methodology will allow tests to analyze the temporal aspects of systems.

The concepts that we introduce in this paper can be useful for testing other models of timed systems. For example, the definitions can be easily adapted to timed automata [Alur and Dill, 1994]. Other definitions of timed I/O automata (e.g. [Higashino et al., 1999; Springintveld et al., 2001]) are restricted to deterministic behavior, while we are more expressive since we allow nondeterminism in specifications and (partially) in implementations. Regarding testing for timed systems, some proposals have already appeared in the literature (e.g. [Clarke and Lee, 1997; Higashino et al., 1999; Springintveld et al., 2001]). Our proposal is inspired in [Núñez and Rodríguez, 2003] and differs from the previous ones in several points, mainly because the treatment of time is different. We do not have a notion of clock(s) together with time constraints; we associate time to the execution of actions (the time that it takes for a system to perform and action).

The rest of the paper is organized as follows. In Section 2 we define our language in terms of processes and systems. In Section 3 we show how test cases are defined and describe how to apply them to implementations. Finally, in Section 4 we present our conclusions and some directions for further research.

2. Formal Specification Model: Timed EFSM

In this section we introduce our timed model for specifying systems where temporal constraints play a relevant role. The main difference with respect to usual EFSMs consists in the addition of *time*. Constraints on the time consumed by actions will be given by means of *intervals*. If the action a is attached with the temporal requirement (t_1, t_2) then the execution of a should take at least t_1 units of time and at most t_2 units. Intervals like $(0, t_2)$, (t_1, ∞), or $(0, \infty)$ denote the absence of any temporal lower/upper bound and the absence of any bound, respectively. Variables will be introduced as a way to add some expressivity to classical finite state machines and as communication mechanism between components of the system. For instance, a variable x can be used as a counter that is incremented every time the action b is performed. Then, if b is enabled only if x is less than or equal to 5, then b will be executed at most

5 times. In our model, variables affect temporal requirements as well. For example, we can specify that the time consumed by b is given by a value that is double than that of variable x. Then, the first execution of b must take 2 units of time, the next one 4 units, and so on. Next we introduce the definition of Timed EFSM. We suppose that the number of different variables is equal to m.

DEFINITION 1 A *Timed Extended Finite State Machine*, in short TEFSM, is a tuple $M = (S, I, O, Tr, s_{in}, \bar{y})$ where S is a finite set of states, I is the set of input actions, O is the set of output actions, Tr is the set of transitions, s_{in} is the initial state, and $\bar{y} \in \mathbf{R}^m$ is a tuple of variables.

Each transition $\delta \in Tr$ is a tuple $\delta = (s, s', i, o, Q, Z, C)$ where $s, s' \in S$ are the initial and final states of the transition, $i \in I$ and $o \in O$ are the input and output actions, respectively, associated with the transition, $Q : \mathbf{R}^m \longrightarrow \texttt{Bool}$ is a predicate on the set of variables, $Z : \mathbf{R}^m \longrightarrow \mathbf{R}^m$ is a transformation over the current variables, and $C : \mathbf{R}^m \longrightarrow (\mathbf{R}_+ \cup \{\infty\}) \times (\mathbf{R}_+ \cup \{\infty\})$ returns, for a given value of the variables, a time interval. Intervals of the form (t, t) where the lower and upper bounds coincide will be written just as t.

A *configuration* in M is a pair $(s, \bar{x})$ where $s \in S$ is the current state and $\bar{x}$ is the current tuple of variable values.

We say that $\sigma = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ is a timed *trace* of M if there exist transitions $\delta_1, \ldots, \delta_r \in Tr$ such that $\delta_1 = (s, s_1, i_1, o_1, Q_1, Z_1, C_1)$, ..., $\delta_r = (s_{r-1}, s', i_r, o_r, Q_r, Z_r, C_r)$, the predicate Q is defined such that it holds $Q(\bar{x}) = (Q_1(\bar{x}) \wedge Q_2(Z_1(\bar{x})) \wedge \ldots \wedge Q_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots)))$, the transformation Z is defined as $Z(\bar{x}) = Z_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots))$, and C is defined as $C(\bar{x}) = C_1(\bar{x}) + C_2(Z_1(\bar{x})) + \cdots + C_r(Z_{r-1}(\ldots(Z_1(\bar{x}))\ldots))$. Finally, the addition over intervals is defined as follows: $(a_1, a_2) + (b_1, b_2) = (a_1 + b_1, a_2 + b_2)$. □

Intuitively, for a configuration $(s, \bar{x})$, a transition $\delta = (s, s', i, o, Q, Z, C)$ indicates that if the machine is in state s, receives the input i, and the predicate Q holds for $\bar{x}$, then after a time belonging to the interval $(t_1, t_2) = C(\bar{x})$ (assuming $t_2 \geq t_1$ and $t_1 \neq \infty$) the machine emits the output o and the values of the variables are transformed according to Z. Let us note that (the value of) variables may disable the execution of actions not only by the effect of the function Q but also by the function C. Traces are defined as a sequence of transitions. In this case, the predicate, the transformation function, and the time associated with the trace are computed from the corresponding to each transition belonging to the sequence. Finally, we suppose that addition in real numbers is extended in the following way: $\infty + r = \infty$.

2.1 Composition of components to specify systems

A system is made of several components. Some of the actions performed by these components will be hidden indicating that they are not visible, that

is, they can be neither controlled nor observed. In order to facilitate the understanding, we do not define a *compressed* version of systems where internal communications are omitted yet. So, we generate the whole graph (including both internal and external actions) and then we *delete* internal communications (getting what we call a *simplified system*). Alternatively, we will also present a method to *directly* create the corresponding simplified system.

DEFINITION 2 Let $M_1, \ldots, M_n$ be TEFSMs where for all $1 \leq i \leq n$ we have $M_i = (S_i, I_i, O_i, Tr_i, s_{in_i}, \bar{y}_i)$. Let $I \subseteq \bigcup_i I_i$ and $O \subseteq \bigcup_i O_i$, such that $I \cap O = \emptyset$. The *system* created by the composition of $M_1, \ldots, M_n$ with respect to the actions sets I and O, denoted by $\mathtt{Sys}(M_1, \ldots, M_n, I, O)$, is defined as the TEFSM $M = (S, I, O, Tr, s_{in}, \bar{y})$ where:

- The initial state s_{in} is defined as $s_{in} = (s_{in_1}, \ldots, s_{in_n})$.
- The initial tuple of (tuples of) variable values is $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_n)$.
- $S = S_1 \times \cdots \times S_n$. Actually, it is enough to consider those states reachable from s_{in} by sequences of transitions belonging to Tr.
- Let $s = (s_1, \ldots, s_j, \ldots, s_n)$ and $s' = (s_1, \ldots, s'_j, \ldots, s_n)$ be two states. Then, $(s_j, s'_j, i, o, Q_j, Z_j, C_j) \in Tr_j$ implies $(s, s', i, o, Q, Z, C) \in Tr$, where $Q(\bar{x}_1, \ldots, \bar{x}_n) \equiv Q_j(\bar{x}_j)$, $C(\bar{x}_1, \ldots, \bar{x}_n) = C_j(\bar{x}_j)$ and we have $Z(\bar{x}_1, \ldots, \bar{x}_n) = (\bar{x}_1, \ldots, Z_j(\bar{x}_j), \ldots, \bar{x}_n)$.

Let $\delta = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ be a trace for the TEFSM previously defined. We say that δ is a *chained trace* if $o_r \in O$, $i_1 \in I$, and for all $2 \leq l \leq r$ we have $i_l \notin I \cup O$ and $i_l = o_{l-1}$. □

Let us note that actions not belonging to $I \cup O$ are considered as internal. A chained trace consists of an external input action, a consecutive sequence of paired output/input actions, and finally an external output action. Chained traces are the basis to define our simplified systems. In order to abstract internal computations, systems are transformed into *simplified systems*. The idea is that transitions of a simplified systems are those chained traces belonging to the original system.

DEFINITION 3 Let $Comp = (S, I, O, Tr, s_{in}, \bar{y})$ be a system. We say that $M' = (S', I, O, Tr', s_{in}, \bar{y})$ is the *simplified system* associated with $Comp$, denoted by $\mathtt{Simp}(Comp)$, if S' and Tr' fulfill the following recursive definition:

- $s_{in} \in S'$, and
- If $s \in S'$ and $\delta = (s, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ is a chained trace of $Comp$ then $s' \in S'$ and $(s, s', i_1, o_r, Q, Z, C) \in Tr'$.

We say that $i_1/o_1, \ldots, i_r/o_r$ is a *non-timed evolution*, or simply *evolution*, of M' if there exists a trace $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ of M' such that $Q(\bar{y})$ holds. Given a simplified system $Scomp$, $\texttt{NTEvol}(Scomp)$ denotes the set of non-timed evolutions of $Scomp$.

We say that $((i_1/o_1, \ldots, i_r/o_r), (t_1, t_2))$ is a *timed evolution* of M' if there exists a trace $(s_{in}, s', (i_1/o_1, \ldots, i_r/o_r), Q, Z, C)$ of M' such that $Q(\bar{y})$ holds and $(t_1, t_2) = C(\bar{y})$. Given a simplified system $Scomp$, $\texttt{TEvol}(Scomp)$ denotes the set of timed evolutions of $Scomp$. □

As we said before, a chained trace is converted into a single transition. Then, an evolution is a trace from the initial state of the simplified system. Let us note that all the actions appearing in evolutions are visible (as internal actions are removed by considering transitions formed from chained traces).

2.2 Alternative definition of simplified systems

In this section we present a direct way to define simplified systems without computing the corresponding auxiliary systems. Intuitively, we compute the states of the composition only when they are needed to define the rest of the composite system. In fact, if we are interested only in a part of the whole systems, we could use this method to generate *on the fly* only that portion of the system. For example, one may be interested in the system reachable from a given state s such that all the possible sequences of transitions have labels belonging to some certain sets of inputs and outputs.

DEFINITION 4 Let $M_1, \ldots, M_n$ be TEFSMs with $M_i = (S_i, I_i, O_i, T_i, s_{in_i}, \bar{y}_i)$. We write $(s_1, \ldots, s_j, \ldots, s_n) \xRightarrow{j,\delta,i,o} (s_1, \ldots, s'_j, \ldots, s_n)$ if there exists j such that $\delta = (s_j, s'_j, i, o, Q, Z, C) \in T_j$.

Let $I \subseteq \bigcup_i I_i$ and $O \subseteq \bigcup_i O_i$ be set of actions such that $I \cap O = \emptyset$. The *simplified system* created by the composition of the systems $M_1, \ldots, M_n$ with respect to the set of actions I and O, denoted by $\texttt{SimpSys}(M_1, \ldots, M_n, I, O)$, is given by the tuple $(S, I, O, T, s_{in}, \bar{y})$, where:

- The initial state s_{in} is defined as $s_{in} = (s_{in_1}, \ldots, s_{in_n})$.
- The initial tuple of (tuples of) variable values is $\bar{y} = (\bar{y}_1, \ldots, \bar{y}_n)$.
- $S \subseteq S_1 \times \cdots \times S_n$ is defined as the set of states belonging to T and reachable from s_{in}.
- A transition (v, w, i, o, Q, Z, C) between the states $v = (v_1, \ldots, v_n)$ and $w = (w_1, \ldots, w_n)$ belongs to T if there exist $i_1 \ldots, i_r, o_1, \ldots, o_r$ such that:
  - $o = o_r \in O$,

- $i = i_1 \in I$,
- for all $2 \leq l \leq r$ we have $i_l \notin I \cup O$ and $i_l = o_{l-1}$,
- there exist $\delta_1, \ldots, \delta_r$, $j_1, \ldots, j_r$ and $s_1, \ldots, s_{r-1}$, where for all $1 \leq l \leq r-1$, $s_l = (s_{l1}, \ldots, s_{ln})$, such that

$$v \xrightarrow{j_1,\delta_1,i_1,o_1} s_1 \xrightarrow{j_2,\delta_2,i_2,o_2} \ldots s_{r-1} \xrightarrow{j_r,\delta_r,i_r,o_r} w$$

  where for all $1 \leq l \leq r$, $\delta_l = (s_{lj_l}, s_{l+1j_l}, i_l, o_l, \breve{Q}_l, \breve{Z}_l, \breve{C}_l)$,
- $Q(\bar{x}) \equiv (Q_1(\bar{x}) \wedge Q_2(Z_1(\bar{x})) \wedge \ldots \wedge Q_n(Z_{n-1}(\ldots(Z_1(\bar{x}))\ldots)))$.
- $Z(\bar{x}) = Z_n(Z_{n-1}(\ldots(Z_1(\bar{x}))\ldots))$.
- $C(\bar{x}) = C_1(\bar{x}) + C_2(Z_1(\bar{x})) + \ldots + C_n(Z_{n-1}(\ldots(Z_1(\bar{x}))\ldots))$.

  In the previous three items, for all $1 \leq l \leq r$ we have:

$$\begin{array}{rcl} Q_l(\bar{x}_1, \ldots, \bar{x}_n) & \equiv & (\breve{Q}_l(\bar{x}_{j_l})) \\ Z_l(\bar{x}_1, \ldots, \bar{x}_n) & = & (\bar{x}_1, \ldots, \breve{Z}_l(\bar{x}_{j_l}), \ldots, \bar{x}_n) \\ C_l(\bar{x}_1, \ldots, \bar{x}_n) & = & \breve{C}_l(\bar{x}_{j_l}) \end{array}$$

□

The next result states that simplified systems defined by applying either the Definitions 2 and 3, or the previous definition coincide. The proof of the result follows by taking into account that the way transitions are defined in both Definitions 3 and 4 generate the corresponding states of the system as well as the transitions between them in the same way. Specifically, in both cases the states of the simplified system consist of the tuple of initial states of all TEFSMs as well as any tuple of states that can be reached by chained traces or sequences of chained traces from the initial state.

LEMMA 5 Let $M_1, \ldots, M_n$ be TEFSMs where each M_i is given by $M_i = (S_i, I_i, O_i, Tr_i, s_{in_i}, \bar{y}_i)$, and let $I \subseteq \bigcup_i I_i$ and $O \subseteq \bigcup_i O_i$ be such that $I \cap O = \emptyset$. Then, we have that

$$\texttt{Simp}(\texttt{Sys}(P_1, \ldots, P_n, I, O)) = \texttt{SimpSys}(P_1, \ldots, P_n, I, O)$$

□

3. Definition and Application of Test Cases

A test represents a sequence of inputs applied to the implementation. Once an output is received, we check whether it is the expected one or not. In the latter case, a fail signal is produced. In the former case, either a pass signal is emitted (indicating successful termination) or the testing process continues by applying another input. If we are testing an implementation with input and

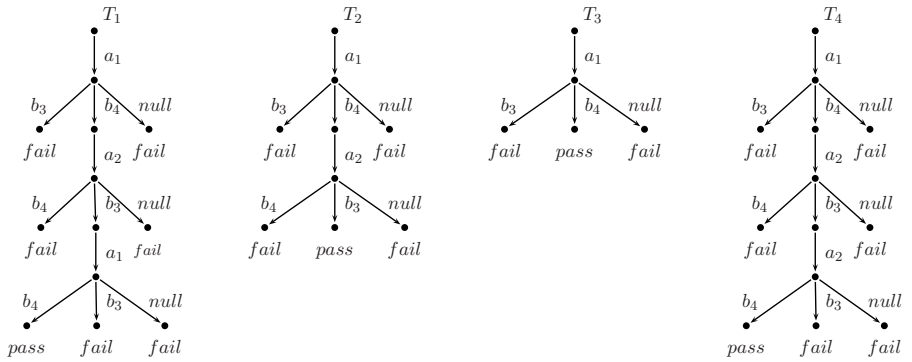

Figure 1. Examples of Test Cases.

output sets I and O, respectively, tests are deterministic acyclic I/O labelled transition systems (i.e. trees) with a strict alternation between an input action and the set of output actions. After an output action we may find either a leaf or another input action. Leaves can be labelled either by *pass* or by *fail*. In the first case we add a *time stamp*, an interval that will be contrasted with the time that the implementation took to arrive to that point. We also add a time stamp in input states to record the time that the performance of the preceding output action took.

DEFINITION 6 A *test case* is a tuple $T = (S, I, O, Tr, s_0, S_I, S_O, S_F, S_P, C)$ where S is the set of states, I and O are disjoint sets of input and output actions, respectively, $Tr \subseteq S \times I \cup O \times S$ is the transition relation, $s_0 \in S$ is the initial state, and the sets $S_I, S_O, S_F, S_P \subseteq S$ are a partition of S. The transition relation and the sets of states fulfill the following conditions:

- S_I is the set of *input* states. We have that $s_0 \in S_I$. For any input state $s \in S_I$ there exists a unique outgoing transition $(s, a, s') \in Tr$. For this transition we have that $a \in I$ and $s' \in S_O$.
- S_O is the set of *output* states. For any output state $s \in S_O$ we have that for any $o \in O$ there exists a unique state s' such that $(s, o, s') \in Tr$. In this case, $s' \notin S_O$. Moreover, there do not exist $i \in I, s' \in S$ such that $(s, i, s') \in Tr$.
- S_F and S_P are the sets of *fail* and *pass* states, respectively. We say that these states are *terminal*. Besides, for any state $s \in S_F \cup S_P$ we have that there do not exist $a \in I \cup O$ and $s' \in S$ such that $(s, a, s') \in Tr$.

Finally, $C : S_P \cup S_I \longrightarrow (\mathbb{R}_+ \cup \{\infty\}) \times (\mathbb{R}_+ \cup \{\infty\})$ is a function associating time interval stamps with passing and input states.

Let $\sigma = i_1/o_1, \ldots, i_r/o_r$. We write $T \stackrel{\sigma}{\Longrightarrow} s$ if $s \in S_F \cup S_P$ and there exist states $s_{12}, s_{21}, s_{22}, \ldots s_{r1}, s_{r2} \in S$ such that $\{(s_0, i_1, s_{12}), (s_{r2}, o_r, s)\} \subseteq Tr$, for all $2 \leq j \leq r$ we have $(s_{j1}, i_j, s_{j2}) \in Tr$, and for all $1 \leq j \leq r-1$ we have $(s_{j2}, o_j, s_{(j+1)1}) \in Tr$.

We say that a test case T is an *instance* of the test case T' if they only differ in the associated function C assigning time intervals.

We say that a test case T is *valid* if the graph induced by T is a tree with root at the initial state s_0. □

In Figure 1 we present some examples of test cases (time stamps are omitted). Next we define the application of a tests suite (i.e. a set of tests) to an implementation. We say that the tests suite $\mathcal{T}$ is *passed* if for any test the terminal states reached by the composition of implementation and test belong to the set of *passing* states. Besides, we give timing conditions according to the temporal constraints included in tests.

DEFINITION 7 Let I be a simplified system and T be a valid test. We write $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ if $T \stackrel{\sigma}{\Longrightarrow} s^T$ and $(\sigma, t) \in \texttt{TEvol}(I)$.

We say that I *passes* the set of tests $\mathcal{T}$, denoted by $\texttt{pass}(I, \mathcal{T})$, if for all test $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ and $\sigma \in \texttt{NTEvol}(I)$ there do not exist s^T and t such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ and $s^T \in S_F$.

We say that I *passes temporally* the set of tests $\mathcal{T}$ if $\texttt{pass}(I, \mathcal{T})$ and for all $(\sigma, t) \in \texttt{TEvol}(I)$ such that $T' \stackrel{\sigma}{\Longrightarrow} s^{T'}$ for some $T' \in \mathcal{T}$, we have that for all $T = (S, I, O, Tr, s, S_I, S_O, S_F, S_P, C) \in \mathcal{T}$ such that $I \parallel T \stackrel{\sigma}{\Longrightarrow}_t s^T$ with $s^T \in S_P \cap S_I$ it holds that $t \in C(s^T)$. □

That is, for passing a test suite we require that (a) no functional behavior that is forbidden by any test is produced by the system and (b) any temporal behavior produced by the system is accepted by all the tests belonging to the test suite.

4. Conclusions and Future Work

In this paper we have presented a model that is suitable for defining the temporal requirements to specify embedded systems. Besides, we have defined how to test an embedded system to find out whether these requirements are fulfilled. The formalism proposed is an extension of the classical EFSMs where actions are provided with the possibility of expressing temporal constraints. These constraints allow to impose maximal and/or minimal bounds for the time consumed by an action as well as strict temporal requirements. The temporal behavior of a system is automatically inferred from the temporal behavior of each component of the system. In particular, temporal constraints on external actions are computed from the temporal constraints on the internal actions participating in the execution of that external action. Tests are sequences of input

actions that stimulate the system and record the outputs produced in response. After each output is received, the test checks whether the time consumed by the trace produced so far fits into the constraint defined by the test.

We are developing temporal implementation relations for embedded systems that define the conditions required for an implementation to *conform* to a specification. In this line, we will take as starting point our previous work in [Núñez and Rodríguez, 2002]. Besides, we are constructing a test derivation algorithm to produce *sound* and *complete* test suites according to these relations, that is, the implementation conforms to the specification if and only if the implementation passes all the tests belonging to the test suite derived from the specification. In this case, we will try to use our recent work on testing probabilistic systems [López et al., 2005] where such a derivation algorithm, although for a probabilistic model, is presented.

References

Alur, R. and Dill, D. (1994). A theory of timed automata. *Theoretical Computer Science*, 126:183–235.

Clarke, D. and Lee, I. (1997). Automatic generation of tests for timing constraints from requirements. In *3rd Workshop on Object-Oriented Real-Time Dependable Systems*.

Higashino, T., Nakata, A., Taniguchi, K., and Cavalli, A. (1999). Generating test cases for a timed I/O automaton model. In *12th Workshop on Testing of Communicating Systems*, pages 197–214. Kluwer Academic Publishers.

Lee, D. and Yannakakis, M. (1996). Principles and methods of testing finite state machines: A survey. *Proceedings of the IEEE*, 84(8):1090–1123.

Lima, L. and Cavalli, A. (1997). A pragmatic approach to generating tests sequences for embedded systems. In *10th Workshop on Testing of Communicating Systems*, pages 288–307. Chapman & Hall.

López, N., Núñez, M., and Rodríguez, I. (2005). Testing of symbolic-probabilistic systems. In *4th Int. Workshop on Formal Approaches to Testing of Software (FATES 2004), LNCS 3395*, pages 49–63. Springer.

Myers, G. (1979). *The Art of Software Testing*. John Wiley and Sons.

Núñez, M. and Rodríguez, I. (2002). Encoding `PAMR` into (timed) `EFSMs`. In *FORTE 2002, LNCS 2529*, pages 1–16. Springer.

Núñez, M. and Rodríguez, I. (2003). Towards testing stochastic timed systems. In *FORTE 2003, LNCS 2767*, pages 335–350. Springer.

Petrenko, A., Yevtushenko, N., and Bochmann, G. v. (1996). Fault models for testing in context. In *Formal Description Techniques for Distributed Systems and Communication Protocols (IX), and Protocol Specification, Testing, and Verification (XVI)*, pages 163–178. Chapman & Hall.

Springintveld, J., Vaandrager, F., and D'Argenio, P. (2001). Testing timed automata. *Theoretical Computer Science*, 254(1-2):225–257.