Model Checking Real-time Component Based Systems with Blackbox Testing

Dang Van Hung and Bui Vu Anh
United Nations University, International Institute for Software Technology
P. O. Box 3058, Macau
e-mail: {dvh,bva}@iist.unu.edu

Abstract

In this paper we propose a simple model for component based real-time systems using duration automata. For this simple model we propose an algorithm for solving the emptiness problem using black-box testing for components which is in the same complexity class as for solving the emptiness problem for untimed component based systems. Furthermore, the verification of behavioural real-time properties in this model can be done with techniques from Duration Calculus.

Keywords: *Component Software, Duration Automata, Automatic Verification, Real-time Systems.*

1. Introduction

The component-based system development supports software reuse and compositionality, hence can reduce the cost for the products. In component based software development, the architectural design of the system plays a key role in achieving the correctness of the system. Architectures include not only the structure of the system but also the behaviour and non-functional aspects of the system. Many models for component based systems have been proposed in the literature [5, 4]. However, those models mainly support the system specifications and understanding, not the verification.

Very often the embedded systems have a simple structure, but complicated real-time behaviours. The architectural design for embedded systems often relies on a minimum specification of component interfaces only, without accessing to the internal behaviour of components. In this paper we propose a simple model for component based real-time systems using duration automata. A duration automaton does not have clock variable like timed automata [1], but has a simple upper bound and lower bound for each transition. It has been shown that many duration properties of real-time systems can be verified automatically for this model. We define a component based real-time system to consist of one host which is a general duration automaton and several components which are duration automata with some restrictions. Components can communicate with their host only. For this model we propose an algorithm for solving the emptiness problem, which plays the key role for checking the safety of the system, using black-box testing for components with a complexity in the same complexity class as for solving the emptiness problem for untimed component based systems.

2. Component Based Real-time System Model

Interface automata were introduced in [2] for specifying component interfaces. We extend interface automata to interface duration automata by associating to each action a simple constraint on action duration in the form of time interval. Let $\mathbb{R}$ be the set of non negative real numbers, and *Intv* be the set of time intervals, *Intv* $\widehat{=} \{[a, b] \mid \tau_1 \in \mathbb{R}, \tau_2 \in \mathbb{R} \cup \{\infty\}\}$.

Definition 1 *An interface duration automaton is a tuple* $M = \langle S, \Sigma, \Delta, \nabla, q, R, F\rangle$, *where*

1. S *is a finite set of states,*
2. Σ, Δ *and* ∇ *are pairwise disjoint alphabets of internal, input and output actions respectively,*
3. $q \in Q$ *is initial state of* M,
4. $R \subseteq S \times (\Sigma \cup \Delta \cup \nabla) \times$ *Intv* $\times S$ *is timed transition relation, and*
5. $F \subseteq S$ *is set of final states.*

For simplicity, for a duration interface automaton M, we will use $S(M)$, $\Sigma(M)$, $\Delta(M)$, $\nabla(M)$, $R(M)$, $q(M)$ and $F(M)$ to denote the corresponding components of M as in the above definition. The untimed version of M, denoted by $untimed(M)$ is the untimed automaton defined in the same way as M except that the transition relation is defined by $untimed(R) \mathrel{\widehat{=}} \{(s, a, s') \mid (s, a, [l, u], s') \in R\}$. Let

$\mathcal{A}(M) \mathrel{\widehat{=}} \Sigma(M) \cup \Delta(M) \cup \nabla(M)$. A configuration of M is a pair $(s, d) \in S \times \mathbb{R}$. A configuration (s, d) says that M has been in state s for d time units. So, the initial configuration of M is $(q, 0)$, and an acceptance configuration of M is a configuration (s, d) where $s \in F$. A transition of M is either a time transition of the form $(s, d) \xrightarrow{\delta} (s, d + \delta)$ ($\delta \in \mathbb{R}$ and $\delta \geq 0$) or a discrete transition of the form $(s, d) \xrightarrow{a} (s', 0)$, where $a \in \Sigma \cup \Delta \cup \nabla$, $(s, a, [l, u], s') \in R$ and $l \leq d \leq u$. In words, a discrete transition can take place only if the amount of time it has been enabled, i.e. staying in the source state, satisfies the time constraint associated to it.

Let $M_1, M_2, \ldots, M_k$ be duration interface automata. They are said to be composable iff $\Delta(M_i) \cap \Delta(M_j) = \nabla(M_i) \cap \nabla(M_j) = \emptyset$ for all $i \neq j$, and $\Sigma(M_i) \cap \mathcal{A}(M_j) = \emptyset$ for all $i \neq j$. A finite set of composable duration interface automata $\mathbf{S} \mathrel{\widehat{=}} \{M_1, M_2, \ldots, M_k\}$ is called a (real-time) system. The components of **S** are running in parallel and communicate with one another synchronously provided their own time constraints are satisfied.

A configuration of system **S** is a tuple $C \mathrel{\widehat{=}} (c_1, c_2, \ldots, c_k)$, where c_i is a configuration of M_i. The configuration of **S** in which c_i is the initial configuration of M_i for all $i \leq k$, is called the initial configuration of **S**. An acceptance configuration of **S** is a configuration in which c_i is an acceptance configuration of M_i for all i. For $a \in \cup_{i=1}^{k} \mathcal{A}(M_i)$, let $dom(a) \mathrel{\widehat{=}} \{i \mid a \in \mathcal{A}(M_i)\}$. We combine a time transition with a following discrete transition into one and define the transition relation of **S** as: $((s_1, d_1), \ldots, (s_k, d_k)) \xrightarrow{(\delta, a)} ((s'_1, d'_1), \ldots, (s'_k, d'_k))$ for $\delta \geq 0$ and $a \in \cup_{i=1}^{k} \mathcal{A}(M_i)$ iff for all $i \in dom(a)$ there is an interval $[l_i, u_i] \in$ *Intv* such that $(s_i, a, [l_i, u_i], s'_i) \in R(M_i)$, $d_i + \delta \in [l_i, u_i]$ and $(s'_i, d'_i) = (s_i, 0)$, and if $i \notin dom(a)$ then $(s'_i, d'_i) = (s_i, d_i + \delta)$.

A path p of **S** is a sequence of consecutive transitions $C_{i-1} \xrightarrow{(\delta_i, a_i)} C_i$, $i = 1, \ldots, n$. A path such that C_0 is the initial configuration of **S** is called a behaviour. We denote a behaviour of **S** by $\sigma \mathrel{\widehat{=}} C_0 \xrightarrow{(\delta_1, a_1)} C_1 \xrightarrow{(\delta_2, a_2)} C_2 \ldots \xrightarrow{(\delta_n, a_n)} C_n$. Let p be a path $C_0 \xrightarrow{(\delta_1, a_1)} C_1 \xrightarrow{(\delta_2, a_2)} C_2 \ldots \xrightarrow{(\delta_n, a_n)} C_n$. A configuration C is reachable from C_0 in d time units on the path p iff there are i and δ satisfying that $i < n \wedge \delta_{i+1} \geq \delta \geq 0$ or $i = n \wedge \delta \geq 0$ such that $C_i \xrightarrow{\delta} C$ and $\sum_{j=1}^{i} \delta = d$.

For an alphabet A, a timed string (word) over A is a sequence $w \mathrel{\widehat{=}} (a_1, t_1)(a_2, t_2) \ldots (a_k, t_k)$, where $a_i \in A$ and $t_i \in \mathbb{R}$ for $i \leq k$, and $0 \leq t_i \leq t_{i+1}$ for $1 \leq i \leq k - 1$. Let $B \subseteq A$. We expand the projection $.|_B$ for strings to a projection for timed strings $.|_B$ as: for a timed string w, $w|_B$ is the subsequence of w consisting of those (a_j, t_j) for which $a_j \in B$.

For a behaviour σ, let $w(\sigma)$ be a timed word defined by $w(\sigma) \mathrel{\widehat{=}} (a_1, t_1)(a_2, t_2) \ldots (a_n, t_n)$, where $t_i \mathrel{\widehat{=}} \sum_{j=1}^{i} \delta_j$. $w(\sigma)$ is called timed word of the system **S** if the last configuration of σ is an acceptance configuration of **S**. Let $\mathcal{L}(\mathbf{S})$ denote the set of timed words of the system **S**.

A subsystem of **S** is a subset of $\{M_1, \ldots, M_k\}$.

Theorem 1 *Let $\mathbf{S}'$ be a subsystem of $\mathbf{S}$. A timed word w over $\mathcal{A}(\mathbf{S})$ is a timed word of $\mathbf{S}$ if and only if $w|_{\mathcal{A}(\mathbf{S}')}$ is a timed word of $\mathbf{S}'$ and $w|_{\mathcal{A}(\mathbf{S}-\mathbf{S}')}$ is a timed word of $\mathbf{S} - \mathbf{S}'$.*

The emptiness problem for a system plays the key role in checking the safety. From the obvious corresponding of component systems and timed automata, it follows that the emptiness of the set of timed words of a system **S** is decidable, but has very high complexity.

We will see in the next section that with some restrictions that are appropriate for modeling component based systems, we can solve the problem with much lower complexity.

Now, we extend the system model in [3] for untimed component based systems to model component-based real-time systems. The advantage of this model is its simplicity and ability of verifying some properties of a system with not much information from the used components.

Real-time components will be modeled by duration interface automata with some restrictions. The restrictions come from the way the developers are using components. We assume that there is a special input action “reset” which causes the component to return to its initial state. If the component accepts an input at a state s then it is its up to environment to decide when to send the input. Therefore we assume that there is no time constraint for input actions, i.e. in any input transition $(s, a, [l, u], s')$ satisfies that $l = 0$ and $u = \infty$. Like for the model in [3], we also assume the input determinism and output determinism for components for more predictable behaviours.

Definition 2 *A component is a duration automaton $X = \langle S, \Sigma, \Delta, \nabla, q, R, F\rangle$ that satisfies the following conditions:*

1. *$\Sigma = \emptyset$ and $reset \in \Delta$ (no “explicit” internal action),*
2. *$(s, a, [l, u], s') \in R \wedge a \in \Delta$ implies $l = 0 \wedge u = \infty$,*
3. *$(s, reset, [0, \infty), q) \in R$ for all $s \in S$,*
4. *$((s, a, [l, u], s') \in R) \wedge (a \in \nabla)$ implies $u = \infty$, i.e. when an output is ready, it can be taken at any time afterward.*
5. *(input determinism) for $a \in \Delta$, $(s, a, [0, \infty), s') \in R$ and $(s, a, [0, \infty), s'') \in R$ imply $s'' = s'$,*
6. *(output determinism) for $b \in \nabla$ and $b' \in \nabla \cup (\Delta \setminus \{reset\})$, $(s, b, [l, \infty), s') \in R \wedge (s, b', [l', u'], s'') \in R$ implies $s'' = s'$, $l' = l$, $u' = \infty \wedge b = b'$.*

Since the final states of components play no role in our model according to the way components are used, we assume that for any component X, we have $F(X) = S(X)$, i.e. every state of a component is an acceptance state.

A host is simply a duration interface automaton M.

Definition 3 *A component based real-time system **S** is a system consisting of one host and several components **S** $\widehat{=} \langle M, X_1, \ldots, X_k \rangle$, where M is a host, and $X_1, \ldots, X_k$ are components satisfying $\mathcal{A}(X_i) \cap \mathcal{A}(X_j) = \emptyset$ for $i \neq j$, and $\Delta(M) \cup \nabla(M) = \cup_{i=1}^{k} \mathcal{A}(X_i)$.*

Since the alphabets of components are included in the alphabet of the host, it follows from Theorem 1 that a timed word w of a component based system **S** is also a timed word of the host M.However, the statement in the reverse direction does not necessarily hold in general. We can decide if a timed word of M is also a timed word of the system **S** by testing if we are given a limited specification of each component of S.

Let σ be an accepted behaviour of the host M. The sequence $[\sigma] \widehat{=} (a_1, [l_1, u_1]) \ldots (a_n, [l_n, u_n])$ is called an accepted sequence of transitions of M.

Let r be the number of states of M, and m is the maximal number of states of components X_j, $j \leq k$. Let $untimed(\mathbf{S}) \widehat{=} M \times untimed(X_1) \times \ldots \times untimed(X_k)$ be the synchronised product of the untimed automata corresponding to the host M and the components X_j's. The number of states of $untimed(\mathbf{S})$ is bounded by $r * m^k$, and from the definition of the synchronised products it follows that each transition in $untimed(\mathbf{S})$ is a parallel execution of a communication transition in M and a transition in a component with the same label, or an internal transition in M.

We have the following criterion for the emptiness of real-time component based systems in our model. Let P be the length of the longest path (number of transitions) from the initial state to an acceptance state of M in which any cycle is not repeated more than $r * m^k$ times for each time it is entered.

Theorem 2 *The set of timed words of the real-time component based system **S** is not empty if and only if there is an accepted sequence of transitions of the host M $[\sigma] = (a_1, [l_1, u_1]) \ldots (a_n, [l_n, u_n])$ with the length $n \leq P$ such that for its corresponding untimed word $w \widehat{=} a_1 a_2 \ldots a_n$ the word $w|_{\mathcal{A}}(X_i)$ is accepted by $untimed(X_i)$ for all $i \leq k$ and for all $j = 1, \ldots, n$, if $a_j \in \nabla(X_i)$ for some i then either $u_j + \ldots + u_{h+1} \geq d_j$ or there is a positive cycle on the path from $h+1$ to $j-1$ with the length not greater than $r * m^k$, where h is the largest index less than j such that $a_h \in \mathcal{A}(X_i)$, and $q(X_i) \overset{a_1 \ldots a_h|_{\mathcal{A}(X_i)}}{\longrightarrow} s'$ holds in the automaton $untimed(X_i)$, and d_j is the minimum delay of the unique output action of X_i at state s' with label a_j, i.e. $(s', a_j, [d_j, \infty), s'') \in R(X_i)$.*

Hence, a more efficient algorithm for deciding the emptiness of a component based real-time system than the general algorithm for timed automata can be constructed by searching for an acceptance sequence of the host M with the length not longer than P that satisfies the conditions of Theorem 2. Note that these conditions can be verified by black box testing as presented in the next subsection.

3. Model-Checking Component Based Systems with Black-box Testing

A component is regarded as a black box, and its behaviour can only be determined by observing its input/output sequence with a clock. We assume that when the output action is tested, the lower bound for the delay of the transition is also reported in the result. Our assumptions for black box testing real-time component X are:

(a) Whenever X is sent an input symbol in $\Delta(X)$, it immediately outputs a special symbol (not in ∇) "yes" or "no" to indicate whether the input is accepted or not.

(b) X has a special input symbol (not in $\Delta(X)$) "prob" that always makes X, when its current state is s, execute a unique output transition $(s, b, [d, \infty), s')$ if such action exists (i.e. b and d are observable), and "no" if otherwise. So, send(X,"prob") returns "no" if output transition $(s, b, [d, \infty), s')$ does not exist, and (b, d) otherwise.

The following algorithm describes our black box testing procedure. Let X be a component, $w \in \mathcal{A}(X)^*$, let w^j denote the jth element of w. We also assume that a variable d_X records the value of the minimal delay d of the last output symbol b in w when the black box test on w is successful (d_X is introduced just for serving the purpose of the algorithm for checking the emptiness of component based system presented below).

```
BlackboxTest(X,w)
    send "$reset$" to $X$;
    for($j := 0, j < |w|, j++$)
        if $w^j$ is an input symbol
            if send($X$,$w^j$) = "no" return "no";
        if $w^j$ is an output symbol
            if send($X$,"prob") = $(b, d)$
                if $w^j \neq b$ return "no";
                if $w^j = b$  $d_X := d$;
            if send($X$,"prob") = "no" return "no";
    return "yes"
```

The emptiness of a component based real-time system in our model can be solved by the the following testing procedure. Let for a sequence of transition w, $label(w)$ denote the sequence of the labels corresponding to the sequence w.

Input: Component based system **S** $\widehat{=} \langle M, X_1, \ldots, X_k \rangle$

Output: "Yes" if the set of the timed words of **S** is not empty, "No" otherwise.

Method:

(1) Compute P, the length of the longest path (number of transitions) from the initial state to an acceptance state of M in which any cycle is not repeated more than $r * m^k$ times for each time it is entered, by using a searching technique in the graph of M.
(2) Generate all acceptance sequences of transitions of M with length P in a systematic way (e.g. by breadth first searching);
(3) Checking on-the-fly whether any prefix of a generated sequence satisfies the conditions of Theorem 2. This can be done by:
(i) For each prefix of a generated sequence $w = e_1 e_2 \ldots e_n$, for each $i \leq n$ let $e_i = (s_{i-1}, a_j, [l_i, u_i], s_i)$. For $j \leq k$ let $m_j(w)$ be the largest index of w such that $a_{m_j} \in \mathcal{A}(X_j)$ if it exists, otherwise, let $m_j(w) = 0$. Let $deadline_j(w)$ be $\sum_{h=m_j+1}^{n} u_h$ ($m_j(w)$ and $deadline_j(w)$ can be maintained properly).
(ii) If the label a of e_{n+1} belongs to $\Delta(Xj)$, then if **BlackboxTest**($X_j, label(w)|_{\mathcal{A}(X_j)}$) = "no", we_{n+1} does not satisfy the conditions of Theorem 2. Otherwise, update $w := we_{n+1}$, $m_j(w) := n + 1$, $deadline_j(w) := 0$.
(iii) If the label a of e_{n+1} belongs to $\nabla(Xj)$.
If **BlackboxTest**($X_j, label(w)|_{\mathcal{A}(X_j)}$) = "yes", let d be the value of d_{X_j}.
(a) If $deadline_j(w) + u_{n+1} < d$: Verify if there is a positive allowable cyclic path between $m_j(w) + 1$ and n. If such path does not exit, then we_{n+1} does not satisfy the conditions of Theorem 2. Otherwise, update $w := we_{n+1}$, $m_j(w) := n + 1$, $deadline_j(w) := 0$.
(b) If $deadline_j(w) + u_{n+1} \geq d$: The conditions of Theorem 2 are satisfied; update $w := we_{n+1}$, $m_j(w) := n + 1$,
$deadline_j(w) := 0$, $deadline_{j'}(w) := deadline_{j'}(w) + u_{n+1}$ for $j' \neq j$.

If **BlackboxTest**($X_j, label(w)|_{\mathcal{A}(X_j)}$) = "no", the conditions of Theorem 2 are not satisfied. (iv) If the label a of e_{n+1} does not belong to $\cup_{j \leq k} \mathcal{A}(Xj)$, update $w := we_{n+1}$, $deadline_j(w) := deadline_j(w) + u_{n+1}$ for $j \leq k$.

(4) If a generated sequence satisfying the conditions of Theorem 2 is found, returns with "Yes". Otherwise, return with "No".

The time complexity for the worst cases of this algorithm is of $O(P^2 * |\mathcal{A}(\mathbf{S})|^{P+1})$, where $|\mathcal{A}(\mathbf{S})|$ is the size of the alphabet of the system **S**. This time complexity does not depend on the size of the constants occurring in the constraints for the transitions.

It is well-known that the reachability and safety problem can be reduced to the emptiness problem, and hence can be solved with the technique in the previous section. Let **S** $\widehat{=} \langle M, X_1, \ldots, X_k \rangle$ be a component based real-time system. Let *Bad* be a subset of the state set of M. We have to check if states in *Bad* are not reachable. Let M' be M with the set of final states being replaced by *Bad*. States in *Bad* are not reachable in **S** iff the set of timed words of the system $\mathbf{S}' = \langle M', X_1, \ldots, X_k \rangle$ is empty.

The host M of a system **S** $\widehat{=} \langle M, X_1, \ldots, X_k \rangle$ is designed to satisfy some real-time requirements. Because M is just a duration interface automaton, it is much easier to verify if M satisfies a real-time property than to do it for a timed automaton. In order to achieve its functionality, M uses services from components X_j, $j \leq k$. However, if the time performance of X_j is low, then **S** may not be implementable. Therefore, the emptiness testing algorithm presented above can be used to decide whether the time performance of X_j's is acceptable for **S**.

4. Conclusion

We have presented a model for component-based real-time systems which has some advantages over the models in the literature. The main advantage is that it supports the black box testing for checking the emptiness with nearly the same cost as for untimed component-based systems. Actually, from the simplicity of the proposed architecture of systems we can have a lower complexity, but this would need a more complicated analysis. We also propose a simple technique for the verification of Real-time properties written as a formula in some real-time logic for our model. We believe that although our model is simple, it is good for the modelling and verification of many embedded real-time systems in practice.

References

[1] R. Alur and D. Dill. A Theory of Timed Automata. *Theoretical Computer Science*, pages 183–235, 1994.

[2] L. de Alfaro and T. A. Henzinger. Interface Automata. In *ACM Symposium on Foundation of Software Engineering (FSE)*, 2001.

[3] Z. D. Gaoyan Xie. Model-checking driven black-box testing algorithms for systems with unspecified components. In *CoRR cs.SE/0404037*, Electronic Edition (link), April 2004.

[4] D. V. Hung. A Model for Component Interfaces for Real-time Systems. Technical Report 296, UNU-IIST, 2004. .

[5] H. Jifeng, Z. Liu, and L. Xiaoshan. Contract-Oriented Component Software Development. Technical Report 276, UNU-IIST, 2003.