The United Nations University

UNU/IIST

International Institute for Software Technology

Model Checking Component Based Systems with Blackbox Testing

Dang Van Hung and Bui Vu Anh

October 2004

UNU-IIST Report No. 317

UNU-IIST and UNU-IIST Reports

UNU-IIST (United Nations University International Institute for Software Technology) is a Research and Training Centre of the United Nations University (UNU). It is based in Macau, and was founded in 1991. It started operations in July 1992. UNU-IIST is jointly funded by the Governor of Macau and the governments of the People's Republic of China and Portugal through a contribution to the UNU Endownment Fund. As well as providing two-thirds of the endownment fund, the Macau authorities also supply UNU-IIST with its office premises and furniture and subsidise fellow accommodation.

The mission of UNU-IIST is to assist developing countries in the application and development of software technology.

UNU-IIST contributes through its programmatic activities:

1. Advanced development projects, in which software techniques supported by tools are applied,
2. Research projects, in which new techniques for software development are investigated,
3. Curriculum development projects, in which courses of software technology for universities in developing countries are developed,
4. University development projects, which complement the curriculum development projects by aiming to strengthen all aspects of computer science teaching in universities in developing countries,
5. Schools and Courses, which typically teach advanced software development techniques,
6. Events, in which conferences and workshops are organised or supported by UNU-IIST, and
7. Dissemination, in which UNU-IIST regularly distributes to developing countries information on international progress of software technology.

Fellows, who are young scientists and engineers from developing countries, are invited to actively participate in all these projects. By doing the projects they are trained.

At present, the technical focus of UNU-IIST is on formal methods for software development. UNU-IIST is an internationally recognised center in the area of formal methods. However, no software technique is universally applicable. We are prepared to choose complementary techniques for our projects, if necessary.

UNU-IIST produces a report series. Reports are either Research R, Technical T, Compendia C or Administrative A. They are records of UNU-IIST activities and research and development achievements. Many of the reports are also published in conference proceedings and journals.

Please write to UNU-IIST at P.O. Box 3058, Macau or visit UNU-IIST's home page: http://www.iist.unu.edu, if you would like to know more about UNU-IIST and its report series.

Chris George, Acting Director

The United Nations University

UNU/IIST

International Institute for Software Technology

P.O. Box 3058
Macau

Model Checking Component Based Systems with Blackbox Testing

Dang Van Hung and Bui Vu Anh

Abstract

In this report we propose a simple model for component based real-time systems using duration automata. A component based real-time system consists of a host which is a general duration automaton and several components which are duration automata with some restrictions. Components can communicate with the host only. For this model we propose an algorithm for solving the emptiness problem 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. Furthermore, the verification of behavioural real-time properties in this model can be done with techniques from Duration Calculus.

Dang Van Hung is a research fellow for the research project "Theories and Design Methods for Real-time Systems" since October 1995 being on leave of absence from Institute of Information Technology, Nghia Do, Cau Giay, Hanoi, Vietnam. He has a PhD (equivalent) degree in Computer Science (concurrent systems) in 1988, Computer and Automation Research Institute (SZTAKI), Hungarian Academy of Sciences, Budapest, Hungary, and a BSc degree in Mathematics (numerical methods) in 1977, Hanoi University, Hanoi, Vietnam.

His research interests include Formal Techniques of Programming, Concurrent and Distributed Computing, Design Techniques for Real-Time systems.

e-mail: dvh@iist.unu.edu

Bui Vu Anh is a lecturer at the faculty of mathematics, mechanics and informatics, School of Natural Sciences, Hanoi National University. He was a UNU-IIST fellow from 5 January to 4 October 2004.

e-mail: anhbv@vnu.edu.vn

Contents

1 Introduction

Component-based systems become popular because of many advantages. 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 [9, 8]. 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. We believe that a model for component-based real-time systems that supports this kind of design is useful.

In this paper we propose a simple model for component based real-time systems using duration automata. Duration automata were introduced in [3] as a model of real-time systems. 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 system 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 its host only. For this model we propose an algorithm for solving the emptiness problem 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. In this model, each behaviour of a system is also a behavior of its host. Hence the verification of many behavioural real-time properties of the system can be reduced to the one for the host which can be done automatically with techniques from Duration Calculus as said earlier.

The paper is organised as follows. In the next section, we introduce a model of component based system using duration automata network and propose an algorithm for checking the emptiness of a component based system. In Section 3, we discuss about possible applications of the algorithm. Section 4 devotes to an example for illustrating the use of our technique. The last section is the conclusion of the report.

2 Checking Untimed Properties of Real-time Component-based Systems

2.1 Duration Interface Automata

Interface automata were introduced in [4] for specifying component interfaces. An interface automaton has internal, input and output actions. 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. Formally, interface duration automata are defined as follows. Let $\mathbb{R}$ be the set of non negative real numbers, and *Intv* be the set of time intervals, $Intv \mathrel{\widehat{=}} \{[a,b] \mid \tau_1 \in \mathbb{R}, \tau_2 \in \mathbb{R} \cup \{\infty\}\}$.

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 time it has been enabled, i.e. stayed 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 their corresponding input/output alphabets are pairwise disjoint, i.e. $\Delta(M_i) \cap \Delta(M_j) = \nabla(M_i) \cap \nabla(M_j) = \emptyset$ for all $i \neq j$, and their internal actions are local to them, i.e. $\Sigma(M_i) \cap \mathcal{A}(M_j) = \emptyset$ 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. The behaviour of system **S** is described precisely as follows.

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 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 $dom(a) \mathrel{\widehat{=}} \{i \mid a \in \mathcal{A}(M_i)\}$. $dom(a)$ is the set of the indexes of those components which are involved in the action a. 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 $\boldsymbol{S'}$ be a subsystem of $\boldsymbol{S}$. A timed word w over $\mathcal{A}(\boldsymbol{S})$ is a timed word of $\boldsymbol{S}$ if and only if $w|_{\mathcal{A}(\boldsymbol{S'})}$ is a timed word of $\boldsymbol{S'}$ and $w|_{\mathcal{A}(\boldsymbol{S}-\boldsymbol{S'})}$ is a timed word of $\boldsymbol{S} - \boldsymbol{S'}$.*

Proof: By direct check using induction on the length of behaviours. □

Corollary 1 *A timed word w over $\mathcal{A}(\boldsymbol{S})$ is a timed word of $\boldsymbol{S}$ if and only if $w|_{\mathcal{A}(M_i)}$ is a timed word of M_i for all $1 \leq i \leq k$.*

Theorem 2 *The emptiness of the set of timed words of a system $\boldsymbol{S}$ is decidable.*

Proof: Define a timed automaton with k clock variables $T(\mathbf{S})$ by

$$\begin{aligned} T(\mathbf{S}) &= \langle S_1 \times \ldots \times S_k, \\ &\quad \mathcal{A}(\mathbf{S}), \{x_1, \ldots, x_k\}, E, (q_0, \ldots, q_k), F_1 \times \ldots \times F_k \rangle \end{aligned}$$

where $S_1 \times \ldots \times S_k$ is the state set, $\{x_1, \ldots, x_k\}$ is the clock variable set of $T(\mathbf{S})$, E is the set of transitions, $\mathcal{A}(\mathbf{S})$ is the set of transition labels of $T(\mathbf{S})$, $(q_0, \ldots, q_k)$ is the initial state, and $F_1 \times \ldots \times F_k$ is the set of final states of $T(\mathbf{S})$. The transition set E is defined as follows. There is an transition e in E for $T(\mathbf{S})$ with the label a from a state $(s_1, \ldots, s_k)$ to a state $(s'_1, \ldots, s'_k)$ iff for $i \in dom(a)$ there is a transition $e_i = (s_i, a, [l_i, u_i], s'_i)$ in R_i, and for $i \notin dom(i)$ $s_i = s'_i$. Transition e has the clock constraint $\wedge_{i \in dom(a)} l_i \leq x_i \leq u_i$, and resets the clocks in $\{x_i \mid i \in dom(a)\}$. By direct check, it follows that a timed word w is accepted by the system $\mathbf{S}$ if and only if it is accepted by the timed automaton $T(\mathbf{S})$. Because the emptiness of $T(\mathbf{S})$ is decidable, the emptiness of $\mathbf{S}$ is also decidable.

□

When converting the emptiness of systems $\mathbf{S}$ into the emptiness of timed automata, we have ignored the simplicity of the time constraints of $\mathbf{S}$ comparing with timed automata, and use the general algorithm for solving the emptiness of timed automata to decide the emptiness of a system $\mathbf{S}$ which involves a large number of regions and hence has very high complexity. We will see in the next section that with some restrictions that are appropriate for meddling component based systems, we can solve the problem with much lower complexity.

2.2 A Model Component Based Real-time Systems

We extend the system model in [6] 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. In this model, there is a distinction between the host system and components.

Real-time 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. Namely, the details of a component are hidden to the system developers, in particular its internal actions and states are hidden. Therefore, its internal actions can be encoded with its states, and hence, we can assume the absence of the internal actions for the component. We assume that there is a special input action "reset" which causes the component return to its initial state. If the component accepts an input at a state s then it is its 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 [6], 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. $(s, a, [l, u], s') \in R$ *and* $a \in \Delta$ *imply* $l = 0$ *and* $u = \infty$,
3. $(s, reset, [0, \infty), q) \in R$ *for all* $s \in S$,
4. $(s, a, [l, u], s') \in R$ *and* $a \in \nabla$ *imply* $u = \infty$, *i.e. when an output is ready, it can be taken at any time afterward.*
5. *for* $a \in \Delta$, $(s, a, [0, \infty), s') \in R$ *and* $(s, a, [0, \infty), s'') \in R$ *imply* $s'' = s'$ *(input determinism),*
6. *for* $b \in \nabla$ *and* $b' \in \nabla \cup (\Delta \setminus \{reset\})$, *the conditions* $(s, b, [l, \infty), s') \in R$ *and* $(s, b', [l', u'], s'') \in R$ *imply* $s'' = s'$, $l' = l$, $u' = \infty$ *and* $b = b'$ *(output determinism).*

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. The restrictions imposed on components leads to the following property for them:

Theorem 3 *Let* X *be a component. A timed word* $w = (a_1, t_1) \dots (a_n, t_n)$ *over* $\mathcal{A}(X)$ *is a timed word of* X *if and only if its corresponding untimed word* $a_1 a_2 \dots a_n$ *is accepted by the untimed automaton untimed*(X), *and for all* $j = 1, \dots, n$, *if* $a_j \in \nabla(X)$ *then* $t_j - t_{j-1} \geq u_j$, *where* $s_0 = q(X)$, $t_0 \mathrel{\widehat{=}} 0$, $s_0 \stackrel{a_1 \dots a_j}{\longrightarrow} s_j$ *in the automaton untimed*(X), *and* $(s_{j-1}, a_j, [l_j, u_j], s_j) \in R(X)$.

Proof: By direct check with induction on the length of w. □

Host

A host is simply a duration interface automaton M. There is no restriction on hosts since we want them to be as powerful as possible, and duration interface automata are a restricted model for hosts already. Each internal action of a host could be a computation step or a communication step with its environment.

Component Based Real-time Systems

We are ready to define component based real-time systems. We use a simple structure for the system model so that we can have richer properties, lower complexity for model checking but still can model many practical systems. In this structure, each component is closed in the sense that it communicates only with its host.

Definition 3 *A component based real-time system* $\boldsymbol{S}$ *is a system consisting of one host and several components*

$$\boldsymbol{S} \mathrel{\widehat{=}} \langle M, X_1, \ldots, X_k \rangle$$

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* $\Delta(M) \cup \nabla(M) = \cup_{i=1}^{k} \mathcal{A}(X_i)$*.*

In the figures that appear later in this paper, we will use the CSP style to indicate input and output action. Namely, for an action a of a duration automaton A, the label $!a$ is to indicate action a with $a \in \nabla(A)$, and the label $?a$ is to indicate action a with $a \in \Delta(A)$.

Since the alphabets of components are included in the alphabet of the host, it follows from Corollary 1 that a timed word w of a component based system $\mathbf{S}$ is also a timed word of the host M (because w is the projection of itself on $\mathcal{A}(M)$). 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 $\mathbf{S}$ by testing if we are given a limited specification of each component of S.

Let σ be an accepted behaviour of the host M. By the definition of behaviours of a system of one duration interface automaton, $\sigma = (s_0, 0) \stackrel{(\delta_1, e_1)}{\longrightarrow} (s_1, 0) \ldots \stackrel{(\delta_n, e_n)}{\longrightarrow} (s_n, 0)$ where $s_i \in S(M)$, $s_0 = q(M)$, and $e_i = (s_i, a_i, [l_i, u_i], s_{i+1})$ are transitions of M (here, we use the name of transitions instead of their label in the behaviours without fear of confusion). If we replace δ_i by any δ_i' such that $\delta_i' \in [l_i, u_i]$, we also get a behaviour of M. Hence, we can replace δ_i by $[l_i, u_i]$ in the behaviour σ to express the set of all behaviors corresponding to the sequence of transitions of σ, and call the resulting sequence $[\sigma] \mathrel{\widehat{=}} (a_1, [l_1, u_1]) \ldots (a_n, [l_n, u_n])$ an accepted sequence of transitions of M. By Theorem 1, a time word $w = (a_1, t_1) \ldots (a_n, t_n)$ is a timed word of the system $\mathbf{S}$ if and only if

1. $w|_{\mathcal{A}(M)}$ $(= w)$ is a timed word of M, i.e. $t_1, \ldots, t_n$ satisfy that $t_1 \in [l_1, u_1]$, $t_{i+1} - t_i \in [l_{i+1}, u_{i+1}]$, and
2. for all i, the timed word $w|_{\mathcal{A}(X_i)}$ is a timed word of X_i, i.e. $w|_{\mathcal{A}(X_i)}$ satisfies the conditions of Theorem 3.

We formulate this observation in the following theorem which will give a more efficient way to solve the emptiness of the component based real-time system $\mathbf{S}$.

Theorem 4 *A timed word of the component based real-time system* $\boldsymbol{S}$ *exists if and only if there is an accepted sequence of transitions of the host* M $[\sigma] \mathrel{\widehat{=}} (a_1, [l_1, u_1]) \ldots (a_n, [l_n, u_n])$ *such that for its corresponding untimed word* $w \mathrel{\widehat{=}} a_1 a_2 \ldots a_n$*,* $w|_{\mathcal{A}}(X_i)$ *is accepted by the untimed automaton untimed*(X_i) *for all* $i \leq k$*, and the linear constraint system* $\boldsymbol{C}$ *defined below has a solution.* $\boldsymbol{C}$ *is defined as*

1. *for all $j = 1, \ldots, n$, if $a_j \in \nabla(X_i)$ for some i then the constraint $t_j - t_h \geq d_j$ is in $\boldsymbol{C}$, where h is the largest index less than j such that $a_h \in \mathcal{A}(X_i) \cup \{\epsilon\}$, and $q(X_i) \overset{a_1 \ldots a_h|_{\mathcal{A}(X_i)}}{\longrightarrow} s'$ holds in the automaton untimed(X_i), and $(s', a_j, [d_j, \infty), s'') \in R(X_i)$ (the unique output action of X_i at state s' with label a_j), by our convention, $a_0 = \epsilon$ (the empty word) and $t_0 = 0$.*
2. *$t_0 \leq t_1 \leq \ldots \leq t_n$, and $l_j \leq t_j - t_{j-1} \leq u_j$ for $1 \leq j \leq n-1$ are in $\boldsymbol{C}$*
3. *(No other constraints are in $\boldsymbol{C}$).*

In the definition of $\mathbf{C}$ the size of the linear constraint system $\mathbf{C}$ (i.e. the number of constraints) is proportional to the length of w. Since the set of acceptance sequences of M is infinite in general, we cannot test all of them for the conditions of Theorem 4 to decide the emptiness of system S. In the following, we show that we can find a number P such that we have to test only the acceptance sequences of M with length bounded by a fix number P to decide the emptiness of system S.

For the untimed component based model [6], the number P is $r * m^k$, where r is the number of states of M, k is the number of components in S, and m is the maximal number of states of components X_j, $j \leq k$. For real-time component based model P cannot be defined like that.

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
$untimed(\mathbf{S}) \mathrel{\widehat{=}} M \times untimed(X_1) \times \ldots \times untimed(X_k)$ 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. Hence, each accepted sequence of transitions of the host M that satisfies the conditions of Theorem 4 corresponds to exactly one accepted path in the graph representing $untimed(\mathbf{S})$. For convenience we label transitions in $untimed(\mathbf{S})$ by their corresponding transition in M.

Recall that we can use the delay variables δ_i instead of real-time variable t_i in behaviour (path in $untimed(\mathbf{S})$) σ by the substitution $t_i = \sum_{j=1}^{i} \delta_j$. Then the constraints in $\mathbf{C}$ in Theorem 4 become

- $\delta_j + \ldots + \delta_{h+1} \geq d_j$ for j, h and d_j as in the item 1 of Theorem 4, and
- $l_j \leq \delta_j \leq u_j$ for all j.

This constraint system has a solution if and only if

$u_j + \ldots + u_{h+1} \geq d_j$ for j, h and d_j as in Item 1 of Theorem 4.

Here we use the familiar conventions $a + \infty = \infty$ and $\infty \geq a$ for any $a \geq 0$ or $a = \infty$.

By simplifying if necessary, we can assume that all states in $untimed(\mathbf{S})$ are reachable from the initial state. For a constraint T as above, if there is a cycle c in $untimed(\mathbf{S})$ on the subpath ρ between transitions e_{h+1} and e_{j-1} on the path σ in which $u_l > 0$ for some l satisfying $h+1 \leq l \leq j-1$ (called positive cycle), then the constraint T $(u_j + \ldots + u_{h+1} \geq d_j)$, if it has not been satisfied already, can be satisfied by repeating this path for an appropriate number of times. It is interesting to note here that each repetition may add a constraint that has been in $\mathbf{C}$ already, and hence, does not add any new constraint into our constraint system. It is so because that a constraint is on u_j's and d_j's, not on t_j's. So, we can remove constraint T if we know there is a positive cycle in $untimed(\mathbf{S})$ on the path from e_{h+1} and e_{j-1} of σ. Since the number of states of $untimed(\mathbf{S})$ is bounded by $r * m^k$, there must be a cycle on the subpath ρ of σ connecting e_{h+1} and e_{j-1} if the length $|\rho|$ of ρ is greater than $r * m^k$.

Based on this observation, 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 5 *The set of timed words of the real-time component based system $\boldsymbol{S}$ 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 $w \mathrel{\widehat{=}} a_1 a_2 \ldots a_n$ 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. $(s', a_j, [d_j, \infty), s'') \in R(X_i)$.*

Proof:

(a) We have to prove the "only if" part only since the "if" part follows from the above observation. From Theorem 4, we can assume that there is an accepted sequence of transitions of M $[\sigma] \mathrel{\widehat{=}} (a_1, [l_1, u_1]) \ldots (a_n, [l_n, u_n])$ such that its corresponding untimed word $w \mathrel{\widehat{=}} a_1 a_2 \ldots a_n$ satisfies that the projection $w|_{\mathcal{A}}(X_i)$ is accepted by untimed X_i for all $i \leq k$, and such that for all $j = 1, \ldots, n$, if $a_j \in \nabla(X_i)$ for some i then $u_j + \ldots + u_{h+1} \geq d_j$ (1), 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 $(s', a_j, [d_j, \infty), s'') \in R(X_i)$ is the unique output action at state s' of X_i. From the observation mentioned above, $[\sigma]$ corresponds to exactly one accepted path p_σ of the untimed product automaton $untimed(\mathbf{S})$, and satisfies the conditions in Theorem 5 except for the condition $n \leq P$.

If $n \leq P$ is satisfied, then the proof is done. Otherwise, we will prove that we can find a shorter accepted sequence of M that satisfies all the conditions in Theorem 5 but maybe the condition $n \leq P$. For simplicity, we say "a sequence has the property (A)" for "a sequence satisfies all the conditions in Theorem 5 but maybe the condition $n \leq P$". By repeating this process several

times, at last we find an accepted sequence of M that satisfies all the conditions in Theorem 5.

Assume that $n > P$. By the definition of P, p_σ must consist of more than m^k times of repetition of a cycle c in M. There are following cases:

(a) If c does not include any communication action of M then it is also a cycle in $untimed(\mathbf{S})$.

If c is not a positive cycle, by removing the path formed by the repetitions of c, we get a shorter accepted sequence for which has the property (A).

If c is a positive cycle, by removing all repetitions of c, we get an accepted sequence which has the property (A)..

(b) If c includes an communication action of M, then some other components should be involved in the actions in c. Since c is a cycle in M and is repeated more than m^k times, the path formed by this repetition should include at least a repetition a cycle c' in $untimed(\mathbf{S})$.

If c is not a positive cycle, by removing the path formed by one repetition the repetitions of c', we get a shorter accepted sequence which has the property (A).

If c is a positive cycle, by removing the path formed by one repetition the repetitions of c', we get a shorter accepted sequence which has the property (A).

□

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 a the acceptance sequence of the host M with the length not longer than P that satisfies the conditions of Theorem 5. Note that these conditions can be verified by black box testing as presented in the next subsection.

2.3 Unspecified Components and Black-box Testing

From the definition of unspecified components in the previous subsection, a component is regarded as a black box, and its behaviour can only be determined by observing its input/output sequence with a clock. From Theorem 3, in a real-time behaviour $(a_1, t_1) \dots (a_m, t_m)$, the time elements t_j are relevant only for testing whether the constraints on the output are satisfied. Hence we assume that when the output action is tested, the lower bound for the delay of the transition is also reported in the result.

Hence, our assumptions for black box testing real-time component X are:

1. Whenever X is sent an input symbol in $\Delta(X)$, it immediately outputs a special symbol

```
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 ≠ b
                    return "no";
                if w^j = b
                    d_X := d;
            if send(X,"prob") = "no"
                    return "no";
    return "yes"
```

Figure 1: Black Box Test with Timed Constraint

(not in ∇) "yes" or "no" to indicate whether the input is accepted or not.

2. 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 algorithm in Fig. 1 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).

Hence, 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 $\mathbf{S} \mathrel{\widehat{=}} \langle M, X_1, \ldots, X_k \rangle$

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 5. This can be done by:
   (a) For each prefix of a generated sequence $w = e_1e_2 \dots e_n$, 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).
   (b) 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 5. Otherwise, update $w := we_{n+1}$, $m_j(w) := n{+}1$, $deadline_j(w) := 0$.
   (c) 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}.
      i. 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 5. Otherwise, update $w := we_{n+1}$, $m_j(w) := n + 1$, $deadline_j(w) := 0$.
      ii. If $deadline_j(w){+}u_{n+1} \geq d$: The conditions of Theorem 5 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 5 are not satisfied.
   (d) 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 5 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 $\mathbf{S}$. This time complexity does not depend on the size of the constants occurring in the constraints for the transitions. It is interesting that this complexity is in the same class with the complexity for checking the emptiness of untimed component based systems [6] which is much lower than the complexity of the general algorithm for checking the emptiness of timed automata.

3 Model-Checking Untimed and Timed Properties

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. Even the LTL model checking problem is also reducible to the emptiness problem, but we need to define the Büchi acceptance for a system which is not in the scope of this paper. Let $\mathbf{S} \widehat{=} \langle M, X_1, \dots, X_k \rangle$ 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 $\mathbf{S}$ iff the set of timed word of the system $\mathbf{S}' = \langle M', X_1, \ldots, X_k \rangle$ is empty.

The host M of a system $\mathbf{S} \mathrel{\widehat{=}} \langle M, X_1, \ldots, X_k \rangle$ 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 verify if a timed automaton satisfies the same property. 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 $\mathbf{S}$ 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 $\mathbf{S}$. For components X and X', we define $X \leq_{per} X'$ iff all of their components are the same except for their transition sets, and for all $e = (s, a, [l, \infty), s') \in E(X)$ there exists $e = (s, a, [l', \infty), s') \in E(X')$ for which $l \leq l'$, and reversely, for all $e' = (s, a, [l', \infty), s') \in E(X')$ there exists $e = (s, a, [l, \infty), s') \in E(X)$ for which $l \leq l'$. It is obvious from the Theorem 5:

Proposition 1 *Let* $X_j \leq_{per} X'_j$, $j \leq k$, *and* $\boldsymbol{S}' \mathrel{\widehat{=}} \langle M, X'_1, \ldots, X'_k \rangle$. *Then* $\mathcal{L}(\boldsymbol{S}) \subseteq \mathcal{L}(\boldsymbol{S}') \subseteq \mathcal{L}(M)$.

This proposition says that we can replace a component in a component based system in our model with a component with higher performance without violating the feasibility of the system, and any behavioural property (i.e. not depending on the system structure) of M is preserved.

A property written as a Timed CTL (see, e.g. [2]) is not a behavioral property since its satisfaction depends on the structure of automata. Linear Duration Invariants [3] are behavioural properties.

Linear Duration Invariants form a class of important properties of real-time systems. They can be formalised by a class of simple chop-free Duration Calculus formulas of the form $A \leq \ell \leq B \Rightarrow \sum_{s \in Q} c_s \int s \leq T$, where A, B, T and c_s are rationals, B could be ∞. This class was first introduced with the name *linear duration invariants* and investigated in [3]. The duration of a state s is a mapping from time intervals to reals and is denoted by $\int s$. $\int s$, when applied to an observation time interval $[b, e]$ is the accumulated time for the presence of state s over $[b, e]$; and the term ℓ when applied to an observation time interval $[b, e]$ returns the length $e - b$ of the interval. A linear duration invariant $A \leq \ell \leq B \Rightarrow \sum_{s \in Q} c_s \int s \leq T$ simply says that for any observation time interval $[b, e]$, if the length ℓ of the interval satisfies the constraint $A \leq \ell \leq B$ then the durations of the system states over that interval should satisfy the constraint $\sum_{s \in Q} c_s \int s \leq T$.

Each path p of M starting from any reachable configuration $(s_0, 0)$ $(s_0, 0) \stackrel{(\delta_1, a_1)}{\longrightarrow} (s_1, 0) \stackrel{(\delta_2, a_2)}{\longrightarrow} C_2 \ldots \stackrel{(\delta_n, a_n)}{\longrightarrow} (s_n, 0)$ represents an observation of M for which the length of the observation interval is $\sum_{j=1}^{n} \delta_j$, and duration $\int q$ of state q is $\sum_{s_j = q} \delta_j$. M is said to satisfy a linear duration invariant $D \mathrel{\widehat{=}} A \leq \ell \leq B \Rightarrow \sum_{s \in Q} c_s \int s \leq T$ iff any path p starting from a reachable configuration corresponds to an observation for which D is satisfied (in fact, we need to modify

M a bit so that any observation should correspond to a path, but we ignore it here for simplicity, see [3] for more details).

It has been shown in [3, 5] that we can verify if a duration automaton satisfies a linear duration invariant with linear programming techniques with the complexity lower than doing the same for timed automata.

Hence, a model checking technique for checking a linear duration invariant D of $\mathbf{S}$ can be as:

1. Checking if $\mathcal{L}(\mathbf{S})$ is empty using the algorithm in the previous section. If $\mathcal{L}(\mathbf{S})$ is empty, stop with the output "the system is infeasible"
2. Checking if D is satisfied by M using the algorithm in [3]. If D is satisfied, stop with the output "D is satisfied by $\mathbf{S}$". Otherwise, stop with the output "Not sure"

4 Example: Car Navigation System

A Car Navigation System [7] assists the driver of a car to navigate through an area. It has a host (control program) that receives a destination dn from the driver, and then displays a map with route rm on the screen. The system uses three components *GPS*, *Route database* and *Display* (see Fig 2). The component *GPS* (global position system) receives a request qy, and then, if is not reset, will give the current location lo after exactly 4 time units (with support from a satellite). The component *Route database*, if not be reset during its execution, receives a (current) location flo, and a destination tdn in that order, and gives a map mr with route from flo to tdn after exactly 7 time units. The component *Display* receives a command dc for clearing the screen and then a command drm to display the map with route given as the result.

In Fig 2, a transition with no time-interval label does not have time constraint, i.e. it corresponds to the interval $[0, \infty]$.

If we take as the set of final states of the host the set $\{F\}$ (to express that there is a useful computation), then from the procedure in the previous sections, it can be seen that the set of timed words of the system is not empty. However, if for example, the component *Route database* is too slow, says the transition corresponding to the action $!mr$ has the time constraint $[9, \infty]$ (instead of $[7, \infty]$) then the set of timed words of the system is empty, which is not feasible.

Now we want to check a real-time property of the CNS written as a linear duration invariant. We want to express the fact that the time from when the driver give a destination dn to the system, i.e. when the transition $?dn$ takes place, to the time that the system displays a route to dn, i.e. when the transition $!dmr$ takes place, is not more then 15 time unites. There is only one path in the host M from the transition $?dn$ to the transition $!dmr$, and the states occurring on the path are S_2, S_3, S_5, S_6, S_7, S_9 and F. The linear duration invariant $D \mathrel{\widehat{=}} \ell > 15 \Rightarrow \int S_1 + \int S_4 + \int S_8 > 0$ expresses this fact. Indeed, D says that $\int S_1 + \int S_4 + \int S_8 = 0 \Rightarrow \ell \leq 15$,

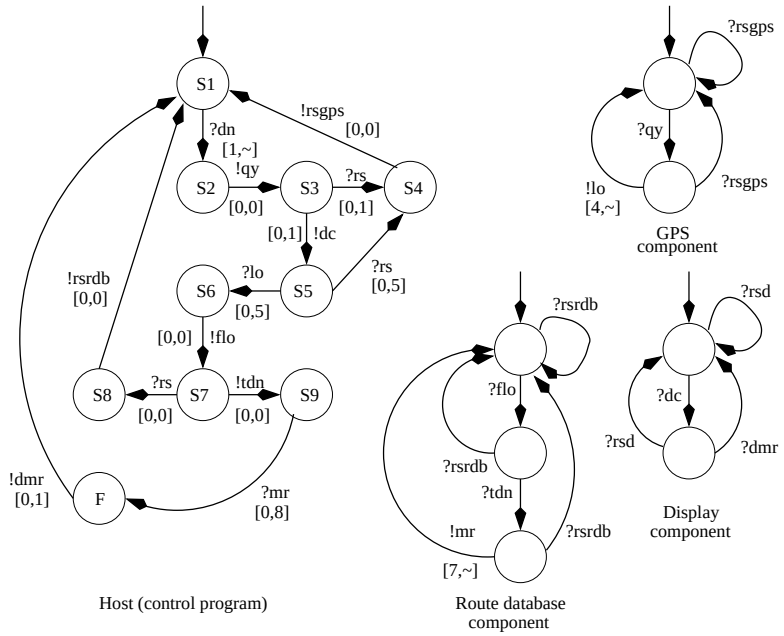

Figure 2: Duration Automata model for CNS

which means that a path in which only states S_2, S_3, S_5, S_6, S_7, S_9 and F can occur corresponds to an observation with the length not longer than 15 time units. Checking if the host M (the control program) satisfies D can be done by a model checker implementing the algorithm in [3].

5 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 though our model is simple, but it is good for modelling and verification of many embedded real-time systems in practice. In our future work, we will enrich our model with some abilities to express the urgency of transitions and to introduce the concurrency and scheduler in the host to increase the expressive power of the model but still can preserve the low complexity.

References

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

[2] B. Berard, M. Bidoit, A. Finkel, F. Laroussinie, A. Petite, L. Petrucci, P. Schoebelen, and P. McKenzie. *Systems and Software Verification, Model-Checking Tecniques and Tools*. Springer-Verlag, 2001.

[3] Z. Chaochen, Z. Jingzhong, Y. Lu, and L. Xiaoshan. Linear Duration Invariants. Research Report 11, UNU-IIST, P.O.Box 3058, Macau, July 1993. Published in: *Formal Techniques in Real-Time and Fault-Tolerant systems*, LNCS 863, 1994.

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

[5] L. X. Dong and D. V. Hung. Checking Linear Duration Invariants by Linear Programming. Research Report 70, UNU-IIST, P.O.Box 3058, Macau, May 1996. Published in Joxan Jaffar and Roland H. C. Yap (Eds.), *Concurrency and Parallelism, Programming, Networking, and Security* LNCS 1179, Springer-Verlag, Dec 1996, pp. 321–332.

[6] 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.

[7] D. K. Hammer. *Software Architectures and Component Technology (Editor: Mehmet Aksit)*, chapter Component-based Architecting for Distributed Real-time Systems. Kluwer, 2002.

[8] D. V. Hung. A Formal Model for Component Interfaces for Real-time Systems. Technical Report 296, UNU-IIST, P.O.Box 3058, Macau, March 2004.

[9] H. Jifeng, Z. Liu, and L. Xiaoshan. Contract-Oriented Component Software Development. Technical Report 276, UNU-IIST, P.O.Box 3058, Macau, April 2003.