Checking Component-based Embedded Software Designs for Scenario-based Timing Specifications*

Hu Jun, Yu Xiaofeng, Zhang Yan, Zhang Tian, Li Xuandong and Zheng Guoliang

State Key Laboratory of Novel Software Technology
Department of Computer Science and Technology
Nanjing University, Nanjing
Jiangsu, P.R.China 210093
hujun@seg.nju.edu.cn

Abstract. In this paper, for real-time embedded software we consider the problem of checking component-based designs for scenario-based timing specifications. By adding time intervals on the actions, we extend the interface automata for modelling real-time systems. The component-based designs are modelled by real-time interface automaton networks, which includes a set of real-time interface automata synchronized by shared actions, and the scenario-based timing specifications are specified by UML sequence diagrams with a set of boolean expressions. Based on analyzing the compatible integer state space of a real-time interface automaton network and its compatible reachability graph, we develop an algorithm to check the consistency between real-time component-based designs and the scenario-based timing specifications.

Keywords: embedded software designs, real-time systems, model checking, interface automata, UML sequence diagrams.

1 Introduction

For most embedded systems, the system correctness not only depends on functionality but also timeliness. And in recent years, a prevalent trend in the embedded computing area is that systems are more and more dominated by software. The software determines the functionality and quality of systems, hence presents lots of challenges to current software techniques. Using component-based design methodology in embedded software domain have been considered as an effective way for handling complexity, increasing system quality. One of the major goals in developing such systems is the validation that whether the component-based designs satisfy the specified timing specifications.

* Supported by the National Natural Science Foundation of China (No.60425204, No.60233020, and No.60273036), the National Grand Fundamental Research 973 Program of China (No.2002CB312001), and by Jiangsu Province Research Foundation (No.BK2004080).

In this paper, for real-time embedded software we consider the problem of checking the consistency of component-based designs for scenario-based timing specifications. Firstly the scenario-based timing specifications are specified by UML sequence diagrams[1] with a set of timming constraints. Then the interface automata[2] are used as a formal model to describe the dynamic behavior of software components, which is suitable for component-based verification. By introducing time intervals into interface automata, we make an extension for modelling real-time systems. And the component-based designs are modelled by real-time interface automaton networks, which includes a set of real-time interface automata synchronized by shared actions. Through analyzing the compatible integer state space of a real-time interface automaton network and its compatible reachability graph, finally we develop an algorithm for the timing behavior consistency checking problem.

2 UML Sequence Diagrams with Timing Constraints

In component-based system designs, the UML sequence diagrams can be used to describe the component interactions. It focus on the temporal order of the message flow. And sending a message, receiving a message, creating or deleting a component instance are all considered as events. According to [3], the semantics of each sequence diagram can be treated as a visual order sequence, which shows a partially ordered set of events displayed in the notation. Therefore, we can formalize the timing sequence diagrams and its behavior as follows:

Definition 1. A timing sequence diagram is a tuple $D = (C, E, M, F, W, S)$ where

- C is a finite set of components;
- E is a finite set of events;
- M is a finite set of messages. For any message $g \in M$, let $g!$ and $g?$ represent the sending and the receiving for g respectively. For any $e \in E$, it is corresponding to the sending or receiving for a message g, denoted by $\phi(e) = g!$ or $\phi(e) = g?$;
- $F : E \mapsto C$ is labelling function which maps each event $e \in E$ to a component $F(e) \in C$;
- W is a finite set whose elements are of the form (e, e') where e and e' are in E and $e \neq e'$, which represents a visual order displayed in D;
- S is a set of boolean expressions on event names, which represents the timing constraints enforced on D. □

Without loss of generality, let any timing constraint expression be of the form $c_0(t_{e_0} - t_{e_0'}) + c_1(t_{e_1} - t_{e_1'}) + \ldots + c_n(t_{e_n} - t_{e_n'}) \sim b$, where t_{e_0}, $t_{e_0'}$, t_{e_1}, $t_{e_1'}$, $\ldots$, t_{e_n}, $t_{e_n'}$ present the time of the event occurrence respectively and c_0, c_1, $\ldots$, c_n, b are real numbers ($b \neq \infty$), $\sim \in \{\leq, <\}$.

Definition 2. For any $D = (C, E, M, F, W, S)$, an event sequence $e_0 \hat{} e_1 \hat{} \ldots \hat{} e_m$ is a *trace* of D iff the following conditions hold:

– all events in E occur in the sequence, and each event occurs only once, i.e. $\{e_0, e_1, \ldots, e_m\} = E$ and $e_i \neq e_j$ for any i, j $(i \neq j,\ 0 \leq i, j \leq m)$; and
– $e_0, e_1, \ldots, e_m$ satisfy the visual order defined by W, i.e. for any e_i and e_j, if $(e_i, e_j) \in W$, then $0 \leq i < j \leq m$. □

We use *timed event sequences*, with the form of $(e_0, \tau_0)\hat{}(e_1, \tau_1)\hat{} \ldots \hat{}(e_m, \tau_m)$, to represent the behavior of real-time systems, in which each event e_i occurs with a relative time stamp τ_i. It means that e_0 takes place τ_0 time units after the system starts, then e_1 takes place τ_1 time units after e_0 takes place, so on and so forth, at last e_m takes place τ_m time units after e_{m-1} takes place.

Definition 3. A timed event sequence $\sigma = (e_0, \tau_0)\hat{}(e_1, \tau_1)\hat{} \ldots \hat{}(e_m, \tau_m)$ is a behavior of a timing sequence diagram $D = (C, E, M, F, W, S)$ iff $e_0\hat{}e_1\hat{} \ldots \hat{}e_m$ is a trace of D, and $\tau_0, \tau_1, \ldots, \tau_m$ satisfy the timing constraints described by S, i.e. for any boolean expression $\sum_{i=0}^{n} c_i(t_{f_i} - t_{f_i'}) \sim b$ in S, there is $\sum_{i=0}^{n} c_i \Delta_i \sim b$ where for each i $(0 \leq i \leq n)$, if $f_i = e_j$ and $f_i' = e_k$, then $\Delta_i = \sum_{u=k+1}^{j} \tau_u$ $(j > k)$or $\Delta_i = -\sum_{u=j+1}^{k} \tau_u$ $(j < k)$. □

3 Real-time Interface Automata and Component-based Designs

3.1 Real-time interface automata

Interface automaton[2] is a light-weight formal language to describe the temporal aspects of software component interfaces. Specifically, it's designed to capture effectively both input assumptions and output guarantees about the order of the interactions between component and its environment. To further describe real-time properties, we need to extend the interface automata with timeliness. In the following, a *real-time interface automaton (RIA)* model is introduced, in which for any transition there is a time interval constraint.

Definition 4. A *RIA* is a tuple $P = (V_P, v_P^{Init}, A_P, I_P, \Gamma_P)$ where:

– V_P is a finite set of states, each state $v \in V_P$;
– $v_P^{Init} \in V_P$ is the initial state;
– A_P is the set of all actions, which includes A_P^I, A_P^O and A_P^H, the mutually disjoint set of input, output and internal actions respectively.
– I_P is a finite set of time intervals; each interval has the form of $[x, y]$ where x, y are nonnegative integer numbers ($x \leq y$, y may be ∞);
– $\Gamma_P \subseteq V_P \times A_P \times I_P \times V_P$ is a set of transitions. □

An action $a \in A_P$ is *enabled* at state $v \in V_P$ if there is a transition $(v, a, [x, y], v') \in \Gamma_P$. An interface automaton is *non-input-enabled*; that is, it's not require that $A_P^I(v) = A_P^I$ for all states $v \in V_P$.

We use a *timed state sequence* of the form $v_0 \xrightarrow{a_0,\tau_0} v_1 \xrightarrow{a_1,\tau_1} \ldots \xrightarrow{a_{m-1},\tau_{m-1}} v_m \xrightarrow{a_m,\tau_m} v_{m+1}$ to describe a behavior of a *RIA*, which means that the system

is starting at state v_0, after τ_i time units changing to v_1 through the transition ignited by the action a_0 and staying there for τ_1 time units, and so on. As the same we have mentioned before, each $\tau_i(0 \leq m)$ is a nonnegative integer number representing the relative time stamp.

Definition 5. For a real-time interface automaton $P = (V_P, v_P^{Init}, A_P, I_P, \Gamma_P)$, a timed state sequence $v_0 \overset{a_0,\tau_0}{\longrightarrow} v_1 \overset{a_1,\tau_1}{\longrightarrow} \ldots \overset{a_{n-1},\tau_{n-1}}{\longrightarrow} v_n \overset{a_n,\tau_n}{\longrightarrow} v_{n+1}$ is a behavior of P iff $v_0 = v_P^{Init}$, and for each $i(0 \leq i \leq n)$, there is $(v_i, a_i, [x_i, y_i], v_{i+1}) \in \Gamma_P$, $x_i \leq \tau_i \leq y_i$. □

3.2 Real-time interface automaton networks

We use *real-time interface automaton networks (RIAN)* to model the component-based designs for a real-time embedded software. It consists of a set of real-time interface automaton which represent the dynamic behaviors of software components. Since an input action of one interface automaton may concide with an output action of the other one, these two interface automata will synchronize on such shared actions, asynchronously interleaving on other actions. Those synchronized actions between any two interface automata (P_i, P_j) are denoted by $shared(P_i, P_j) = A_{P_i} \cap A_{P_j} = (A_{P_i}^O \cap A_{P_j}^I) \cup (A_{P_i}^I \cap A_{P_j}^O)$. More details about the composition of interface automata can be refereed to [2].

For solving the verification problem, we give the formalism of a *RIAN* as follows, including the states , actions and transitions.

Definition 6. Let $N = (K, Z)$ be a *RIAN* where $K = \{P_1, P_2, \ldots, P_n\}$, each $P_i = (V_{P_i}, v_{P_i}^{Init}, A_{P_i}, I_{P_i}, \Gamma_{P_i})$, and $Z = \{shared(P_i, P_j) \mid 1 \leq i, j \leq n, i \neq j\}$ is a set of all shared actions. The states and actions are defined as below:

- an *untimed state* $\overline{v}$ of N is in $V_{P_1} \times V_{P_2} \times \ldots \times V_{P_n}$, that is, $\overline{v} = (v_1, v_2, \ldots, v_n)(v_i \in V_{P_i}, 1 \leq i \leq n)$. The *initial untimed state* of N is $\overline{v}_N^{Init} = (v_{P_1}^{Init}, v_{P_2}^{Init}, \ldots, v_{P_n}^{Init})$;
- a *state* u of N is a pair $u = (\overline{v}, c)$ where $\overline{v} = (v_1, v_2, \ldots, v_n)$ is an untimed state of N and $c : \{v_i | 1 \leq i \leq n\} \mapsto \{R^+\} \cup \{0\}$ is called the *clock function* which maps each v_i to a nonnegative real number that indicates for the P_i how long the system has been staying at v_i. The *initial state* of N is $u_N^{Init} = (\overline{v}_N^{Init}, c^{Init})$ where $\overline{v}_N^{Init}$ is the initial untimed state of N and $c^{Init}(v_{P_i}^{Init}) = 0$ for any $i(1 \leq i \leq n)$. The set of states of N is denoted by U_N;
- the set of actions of N is $A_N = A_N^I \cup A_N^O \cup A_N^H$, where the set of input actions is $A_N^I = \left(\bigcup_{1 \leq i \leq n} A_{P_i}^I\right)/Z$, the set of output actions is $A_N^O = \left(\bigcup_{1 \leq i \leq n} A_{P_i}^O\right)/Z$, and the set of internal actions is $A_N^H = \left(\bigcup_{1 \leq i \leq n} A_{P_i}^H\right) \cup Z$.
- the set of time intervals of N is $I_N = \bigcup_{1 \leq i \leq n} I_{P_i}$. □

Definition 7. Let $N = (K, Z)$ be a *RIAN* where $K = \{P_1, P_2, \ldots, P_n\}$ and $P_i = (V_{P_i}, v_{P_i}^{Init}, A_{P_i}, I_{P_i}, \Gamma_{P_i})(1 \leq i \leq n)$, and $u = (\overline{v}, c)$ and $u' = (\overline{v}', c')$ be its states where $\overline{v} = (v_1, v_2, \ldots, v_n)$ and $\overline{v}' = (v_1', v_2', \ldots, v_n')$. Then the system can change from the state u to u' by an transition within a delay d(denoted by $u \overset{a,d}{\longrightarrow} u'$), if one of the following conditions holds:

– for an action $a \notin Z$, there is a transition $(v_k, a, [x_k, y_k], v'_k) \in \Gamma_{P_k}$ $(1 \leq k \leq n)$ satisfying $x_k \leq c(v_k) + d \leq y_k$ and $c'(v'_k) = 0$. At the same time $v_i = v'_i$ and $c'(v_i) = c(v_i) + d$ for any i $(i \neq k, 1 \leq i \leq n)$; or
– for an action $a \in$ *shared*$(P_i, P_j)(1 \leq i, j \leq n, i \neq j)$, there are transitions $(v_i, a!, [x_i, y_i], v'_i) \in \Gamma_{P_i}$ ($a!$ denote a is an output action) and $(v_j, a?, [x_j, y_j], v'_j) \in \Gamma_{P_j}$ ($a?$ denote a is an input action) satisfying $x_i \leq c(v_i) + d \leq y_i$, $x_j \leq c(v_j) + d \leq y_j$, $c'(v'_i) = 0$ and $c'(v'_j) = 0$. At the same time, $v_k = v'_k$ and $c'(v_k) = c(v_k) + d$ for any $k(k \neq i, j, 1 \leq k \leq n)$.

Then a behavior of N is a timed state sequence: $u_0 \stackrel{a_0,d_0}{\longrightarrow} u_1 \stackrel{a_1,d_1}{\longrightarrow} \ldots \stackrel{a_{n-1},d_{n-1}}{\longrightarrow} u_n \stackrel{a_n,d_n}{\longrightarrow} u_{n+1}$ where u_0 is the initial state of N. □

4 Checking Real-time Component-based Designs for Scenario-based Specifications

Now, for a real-time embedded software the component-based designs are modelled by a real-time interface automaton network, and the scenario-based timing specifications are specified by the sequence diagrams with timing constraints. The problem we concern is to check if the real-time interface automata interact according to the scernio specified by the timing sequence diagrams.

Firstly, we need to give the relation between a behavior of the *RIAN* and a trace of the timing sequence diagram. Let $N = (K, Z)$ be a *RIAN*, and $\varrho = u_0 \stackrel{a_0,d_0}{\longrightarrow} u_1 \stackrel{a_1,d_1}{\longrightarrow} \ldots \stackrel{a_{n-1},d_{n-1}}{\longrightarrow} u_n \stackrel{a_n,d_n}{\longrightarrow} u_{n+1}$ be a behavior of N. Then the corresponding timed action sequence of ϱ is $(a_0, d_0)^\frown(a_1, d_1)^\frown \ldots ^\frown(a_n, d_n)$, and those timed internal action (a_i, d_i) (which satisfies $a_i \in Z$) can be replaced by a pair of timed input and output actions $(a_i!, d_i)^\frown(a_i?, 0)$, as a result we can get an timed action sequence which only contains input and output actions. In fact, each input(resp. output) action a can be considered as an input(resp. output) event e, thus a corresponding timed event sequence, with a form of $(e_0, \tau_0)^\frown(e_1, \tau_1)^\frown \ldots ^\frown(e_r, \tau_r)(r \geq n)$, is obtained which is called the *trail* of ϱ. On the other hand, notice that any message appeared in a sequence diagram actually should be one of the shared actions in N, and each event such as sending message or receiving message should be one of the output or input actions of the corresponding component's *RIA*. However, in the formal definition of the sequence diagram $D = (C, E', M, F, W, S)$ an event $e' \in E'$ just represents an event name and the event content is denoted by $\phi(e')$. Thus if we want to compare the events between the trail of ϱ and the trace of D, it need to compare e with $\phi(e')$.

Let ϱ be a behavior of N, σ be the trail of ϱ, and σ_1 be a subsequence of σ of the form $(e_0, \tau_0)^\frown(e_1, \tau_1)^\frown \ldots ^\frown(e_m, \tau_m)$. For a trace of D with the form of $f_0{}^\frown f_1{}^\frown \ldots ^\frown f_n$, if there are $e_{k_0}, e_{k_1}, \ldots, e_{k_n}$ satisfying that (1) $0 = k_0 < k_1 < \ldots < k_n = m$, (2)$\phi(f_i) = e_{k_i}$ for any i $(0 \leq i \leq n)$, and (3)for any i $(0 \leq i \leq n)$, for any $j \neq k_p$ $(0 \leq j \leq m, 0 \leq p \leq n)$, $\phi(f'_i) \neq e_j$, then the subsequence σ_1 is considered as a *projection* of σ over D. In fact, σ_1 shows that an scenario

described by D *occurs* exactly in behavior ϱ of N. Based on above discussion, we give the formal description of the timing consistency verification problem as below:

Definition 8. Let N be a *RIAN*, and $D = (C, E, M, F, W, S)$ be a timing sequence diagram. N *satisfies* D iff the following conditions hold:

- the scenario described by D occurs in a behavior of N; and
- for any behavior of N, the projection of its trail over D, which have a form of $(e_k, \tau_k)\hat{}(e_{k+1}, \tau_{k+1})\hat{} \ldots \hat{}(e_r, \tau_r)$, satisfies the timing constraints of D, i.e. for any boolean expression $\sum_{i=0}^{n} c_i(t_{f_i} - t_{f_i'}) \sim b$ in S, there is $\sum_{i=0}^{n} c_i \Delta_i \sim b$, where for each i $(0 \leq i \leq n)$ if $\phi(f_i) = e_g$ and $\phi(f_i') = e_h (k \leq g, h \leq r)$, then $\Delta_i = \sum_{u=h+1}^{g} \tau_u$ $(g > h)$or $\Delta_i = -\sum_{u=g+1}^{h} \tau_u$ $(g < h)$. □

It's clearly that the state space of *RIAN* is infinite, so the key point is to find a way for verification in a finite manner.

4.1 Constructing an integer reachability graph

We introduce the *integer behavior* as follows. Let $\varrho = u_0 \xrightarrow{a_0,d_0} u_1 \xrightarrow{a_1,d_1} \ldots \xrightarrow{a_{n-1},d_{n-1}} u_n \xrightarrow{a_n,d_n} u_{n+1}$ is a behavior of a *RIAN*. If d_i is an integer for any i $(0 \leq i \leq n)$, the ϱ is called an *integer behavior* . It follows that any state $((v_1, v_2, \ldots, v_m), c)$ occurring in an integer behavior satisfies $c(v_j)$ is an integer for any j $(1 \leq j \leq m)$, which is called *integer state*.

Theorem 1. Let N be a *RIAN*, and $D = (C, E, M, F, W, S)$ be a timing sequence diagram. N satisfies D iff the scenario described by D occurs in an integer behavior of N, and for any integer behavior of N, the projection of its trail over D satisfies the timing constraints of D. □

Then, for above checking problem, we only need to consider the integer behaviors of the *RIAN*.

To deal with the infinite case in some time intervals, we represent a *time bound* of N by $K = \max\{X_{max}, Y_{max}\}$, where $X_{max} = \max\{x_i | [x_i, \infty) \in I_N (0 \leq i \leq |I_N|)\}$ and $Y_{max} = \max\{y_i | [x_i, y_i] \in I_N (x_i \leq y_i, y_i \neq \infty), 0 \leq i \leq |I_N|\}$. Then let $u = (\overline{v}, c)$ and $u' = (\overline{v}', c')$ be the integer states of N where $\overline{v} = (v_1, v_2, \ldots, v_n)$ and $\overline{v}' = (v_1', v_2', \ldots, v_n')$. and define a time-zone relation R_t over U_N as below:

$$R_t = \left\{ < u, u' > \left| \begin{array}{l} \overline{v} = \overline{v}' \text{ and for each } v_i (1 \leq i \leq n), \text{ either} \\ c(v_i) = c'(v_i) \text{ or } c(v_i) > K \wedge c'(v_i) > K \end{array} \right. \right\}.$$

R_t is an equivalent relation over U_N, and the number of the equivalent class is finite. We use $[u]$ to represent the equivalent class, that is,$[u] = \{u' | u R_t u'\}$, and denote by $[N]$ the finite space which comprise of the equivalent classes under R_t.

Theorem 2. Let N be a *RIAN*, and u_1 and u_2 are the integer states such that $u_1 R_t u_2$. Then there is an integer state u_1' such that $u_1 \xrightarrow{a,d} u_1'$ iff there is an integer state u_2' such that $u_2 \xrightarrow{a,d} u_2'$ and $u_1' R_t u_2'$. □

It shows that the set of integer behaviors of $[N]$ are consistent with those of N.

Based on the time-zone equivalent relation, we can construct a reachability graph $G = \{V_G, L_G\}$ for a *RIAN* N as follows, where V_G is a set of nodes and L_G is a set of edges:

1. for the initial state u_N^{Init} of N, $[u_N^{Init}]$ is in the set V_G, which is called the *initial node*;
2. let $[u]$ be in the set V_G, and K be the time bound of N. For an event a, for an integer d $(0 \leq d \leq K+1)$, if $u \xrightarrow{a,d} u'$ then $[u']$ is in V_G, and the edge $[u] \xrightarrow{a,d} [u']$ is in the set L_G. When $a \in A_N^I$ (resp. A_N^O, A_N^H), the edge $[u] \xrightarrow{a,d} [u']$ is called input (resp. output, internal) edge. We use L_G^I,L_G^O and L_G^H to represent the set of input, output and internal edges respectively.

Let $\rho = l_0 \hat{} l_1 \hat{} \ldots \hat{} l_n (l_i : [u_i] \xrightarrow{a,d} [u_{i+1}] \in L_G, 0 \leq i \leq n)$ be a path of G which is an edge sequence starting from the initial node. From its corresponding timed action sequence $(a_0, d_0) \hat{} (a_1, d_1) \hat{} \ldots \hat{} (a_{n-1}, d_{n-1})$, we can replace those (a_i, d_i) satisfying that a_i is a shared action with a pair of input and output actions $(a_i!, d_i) \hat{} (a_i?, 0)$, as the same way mentioned before. Then we can also get a timed event sequence, with a form of $(e_0, \tau_0) \hat{} (e_1, \tau_1) \hat{} \ldots \hat{} (e_s, \tau_s)(s \geq n-1)$, which is called the *trail* of path ρ. Any edge sequence $l_i \hat{} l_{i+1} \hat{} \ldots \hat{} l_{i+k}$ $(0 \leq i \leq n-1-k)$is called a *subpath* of ρ. We denote the trail of a subpath $l_i \hat{} l_{i+1} \hat{} \ldots \hat{} l_j$ by $\sigma(l_i, l_j)(i \leq j)$, the tail of path ρ by σ_ρ or $\sigma(l_0, l_n)$. Thus each label on the edges either contains an input(output) event or a pair of input and output events. It denotes by $\psi(l_k)$ the event pair corresponding to l_k.

4.2 Eliminating the illegal states

Since the interface automaton is *non-input-enabled*, there will be some *illegal states* during the composition of two interface automata. The illegal state represents that one automaton in *RIAN* may produce an output event that is an input event of another automaton, but is not accepted by the latter one on that state. The formal definition is given below:

Definition 9. Let $N = (K, Z)$ be a *RIAN* where $K = \{P_1, P_2, \ldots, P_n\}$ and $P_i = (V_{P_i}, v_{P_i}^{Init}, A_{P_i}, I_{P_i}, \Gamma_{P_i})(1 \leq i \leq n)$,the set of illegal states is shown as:

$$illegal(N) = \left\{ ((v_1, v_2, ., v_n), c) \in N \left| \begin{array}{l} \exists (v_i, v_j)(i \neq j, 1 \leq i, j \leq n), \\ \exists a \in shared(P_i, P_j), \\ \left(\begin{array}{c} a \in A_{P_i}^O(v_i) \wedge a \notin A_{P_j}^I(v_j) \\ \vee \\ a \in A_{P_j}^O(v_j) \wedge a \notin A_{P_i}^I(v_i) \end{array} \right) \end{array} \right. \right\} \square$$

Based on the theorem 2 and the above definition, the following claim can be established: if u is an integer state satisfying $u \in illegal(N)$, then all the integer states in the time-zone equivalent class $[u]$ are illegal. In this case, $[u]$ is called an *illegal time-zone equivalent class*. And we use $illegal(G)$ to represent those

nodes of reachability graph G which corresponds to the set of illegal time-zone equivalent classes of $[N]$.

Based on the reachability graph G, using the same algorithm framework in [2], we can get a set of compatible time-zone equivalent classes of $[N]$ (denoted by $\Pi_{[N]}$). The basic idea of the algorithm is described briefly as follows. Firstly, an operator $OH_{pre} : 2^{V_G} \mapsto 2^{V_G}$ is defined as: for all sets $\Lambda \subseteq V_G$, $OH_{pre}(\Lambda) = \{\mu \in V_G | \exists(\mu \xrightarrow{a,d} \nu) \in L_G^O \cup L_G^H; \nu \in \Lambda\}$. Then the maximum fixpoint of operator OH_{pre} can be computed over reachability graph G by assigning Λ an initial value *illegal (G)*.

If the set $\Pi_{[N]}$ is empty, from the optimistic view of interface automata, it indicates that there are no helpful environments existing which can prevent the *RIAN* from entering those illegal states in the running time. In this case, the result of the verification problem is false. If the $\Pi_{[N]}$ is nonempty, it implies that there are some kinds of helpful environments making the *RIAN* to work normally; then we only need to concern about the behaviors of the compatible time-zone equivalent class space, which is denoted by $com([N])$. That is, for the verification problem, checking the integer behaviors in $com([N])$ is enough.

Theorem 3. Let N be a *RIAN*. N satisfies a timing sequence diagram D iff:

- there is an integer behavior in the $com([N])$, which contains an occurrence of the scenario described by D; and
- for any integer behavior of $com([N])$, the projection of its trail over D satisfies the timing constraints in D. □

Corresponding to $com([N])$, we use the $com(G)$ to represent the *compatible reachability graph*, which only contains the nodes comprised of $\Pi_{[N]}$. Obviously any path in $com(G)$ represents an integer behavior of $com([N])$. However, during the construction of G there is a constraint enforced on $d(0 \leq d \leq K+1)$, and it leads to a result that an integer behavior in $com([N])$ may not correspond to a path in $com(G)$. So, for any path ρ of $com(G)$ and its tail σ_ρ with a form of $(e_0, \tau_0)\hat{}(e_1, \tau_1)\hat{} \ldots \hat{} (e_m, \tau_m)$, we construct a set of $\theta(\sigma_\rho)$ as below:

$$\theta(\sigma_\rho) = \left\{ (e_0, \tau_0')\hat{}(e_1, \tau_1')\hat{} \ldots \hat{}(e_m, \tau_m') \;\middle|\; \begin{array}{l} \text{for any } i(0 \leq i \leq m), \text{ if } \tau_i = K+1, \\ \text{then } \tau_i' \in [K+1, \infty), \text{ else } \tau_i' = \tau_i. \end{array} \right\}.$$

Now it is clear that every timed event sequence in $\theta(\sigma_\rho)$ is the tail of an integer behavior of $com([N])$, and for any timed event sequence σ' which is the trail of an integer behavior of $com([N])$, there is a path ρ' in $com(G)$ such that $\sigma' \in \theta(\sigma_{\rho'})$. So we can solve the problem of checking a *RIAN* for a timing sequence diagram D by checking every path in $com(G)$.

4.3 Checking the consistency

Even in the compatible integer state space, the number of paths in $com(G)$ may be infinite and the length of a path in $com(G)$ may be also infinite, so we have to solve the problem basing on a finite set of finite paths in $com(G)$. For

that purpose, a *projection path* is introduced as follows. For a sequence diagram $D = (C, E, M, F, W, S)$, a path $\rho = l_0 \hat{} l_1 \hat{} \ldots \hat{} l_n$ with a trail $\sigma(l_0, l_n)$ is called a *projection path* if it satisfies the following conditions: (1) the trail $\sigma(l_i, l_n)(0 \leq i \leq n)$ is a projection of $\sigma(l_0, l_n)$ over D,(2) for any $j, k(0 < j < k < i)$, $l_j \neq l_k$; and (3)for any neighbor $j, k(i \leq j < k \leq n, \psi(l_j) \in E, \psi(l_k) \in E)$, $l_g \neq l_h$ $(j < g < h < k)$. If the timed event sequence $\sigma(l_i, l_n)$ satisfies the boolean timing expressions in D, we say that the projection path *satisfies* D. It is clearly that the length of a projection path in $com(G)$ is finite and the number of projection paths in $com(G)$ is also finite.

And we further consider the loops in the $com(G)$. For a subpath $\rho_1 = l_j \hat{} l_{j+1} \hat{} \ldots \hat{} l_k$, where $l_i : [u_i] \xrightarrow{a_i, d_i} [u_{i+1}](j \leq i \leq k)$. If $l_j = l_k$ then we say that ρ_1 is a *loop*, and $d_{j+1} + d_{j+2} + \ldots + d_k$ is the *elapsed time* on ρ_1, denoted by $T(\rho_1)$. If for any $p, q(j \leq p < q < k)$, there is $l_p \neq l_q$, then we say that ρ_1 is a *simple loop*. For a $D = (C, E, M, F, W, S)$, if any $e \in E$ does not occur in the trail $\sigma(l_j, l_k)$ of ρ_1, we say that ρ_1 is a *flat loop* for D.

Let $\rho = l_0 \hat{} l_1 \hat{} \ldots \hat{} l_m$ be a projection path in $com(G)$, if there is a flat loop $\rho'(T(\rho') > 0)$ for D which contains the edge $l_i : [u_i] \xrightarrow{a_i, d_i} [u_{i+1}]$ $(0 \leq i \leq m)$ and d_i occurs in $\Delta_p (0 \leq p \leq n, c_p \Delta_p \geq 0)$ (see Definition 8), then the path ρ is considered as a *flaw projection path* for D. The edge l_i is called a *flaw point* in ρ. Notice that from a flaw projection path, we can always construct a path in $com(G)$ which does not satisfy the timing constraints in D.

Theorem 4. Let N be a *RIAN*, and G be its integer reachability graph. For a timing sequence diagram D, N satisfies D iff:

- there is a projection path in $com(G)$;
- any projection path in $com(G)$ satisfies D; and
- there is not any flaw projection path for D in $com(G)$. □

Based on the above theorem, we can develop an algorithm for the verification problem, which is depicted in Fig.1. The algorithm consists of two phases which are implemented by depth first search. The first search is to get all simple flat loops for D which are used for checking if there is a flaw point in a projection path. The second one is to find out all the projection paths in $com(G)$ and to check them for D. Since there is only a finite number of projection paths in $com(G)$ and the algorithm is based on depth first search method, the complexity of the algorithm is proportional to number and length of projection paths in $com(G)$.

5 Conclusion

Based on the solution in this paper, we are developing a prototype tool. For future work, we will do more case studies in practical use, and consider the resource and energy analysis in the embedded real-time software systems.

```
currentpath := ⟨[u0]⟩; loopset := ∅;
repeat
  node := the last node of currentpath;
  if node has no new successive node then delete the last node of currentpath
  else begin node := a new successive node of node;
          if node is such that there is a simple flat loop for D in the path
            of currentpath^node
          then put the loop into loopset
          else append node to currentpath;
       end
until currentpath = ⟨⟩;
```

```
currentpath := ⟨[u0]⟩; is_projection_path:=false;
repeat
  node := the last node of currentpath;
  if node has no new successive node then delete the last node of currentpath
  else begin node := a new successive node of node;
          if node satisfies that the path currentpath^node is a projection path
          then begin check if the projection path satisfies D;
                     if no, return false;
                     is_projection_path :=true;
                     check if the projection path is a flaw projection path for D;
                     if yes, return false;
               end
          if node satisfies that currentpath^node is a prefix of projection path
          then append node to currentpath;
       end
until currentpath = ⟨⟩;
if is_projection_path then return true else return false.
```

Fig. 1. Algorithm to check a *RIAN* for a timing sequence diagram D

References

1. G. Booch, J. Rumbaugh and I. Jacobson. *The Unified Modeling Language User Guide*, Addison-Wesley, 1999.
2. L. de Alfaro and T. A. Henzinger. Interface Automata. In *Proc. of ESEC/FSE 01*, Austria, 2001
3. Doron A. Peled. *Software Reliability Methods*. pp300-305, Springer, 2001.
4. J. Seemann, J. WvG. Extension of UML Sequence Diagrams for Real-Time Systems. In *Proc. International UML Workshop*, LNCS 1618, pp240-252, 1998.
5. Rajeev Alur and Mihalis Yannakakis. Model Checking of Message Sequence Charts. In *Proc. the 10th International Conference on Concurrency Theory*, LNCS 1664, pp114-129, 1999.
6. Thomas Firley, Michaela Huhn, Karsten Diethers, et al. Timed Sequence Diagrams and Tool-Based Analysis - A Case Study. In *Proc. of the Second International Conference on UML (UML99)*, LNCS 1732, pp.645-660, 1999.
7. L. de Alfaro, T. A. Henzinger, and M. Stoelinga. Timed Interfaces. In *Proc. of EMSOFT 2002*, LNCS 2491,pp108-122, 2002.
8. Alexandre David, M. Oliver, and Wang Yi. Formal Verification of UML Statecharts with Real-Time Extensions. In: *FASE2002*, LNCS 2306, 2002, pp.218-232.
9. Hu Jun, Yu Xiaofeng, Zhang Yan, et al. Scenario-Based Verification for Component-Based Embedded Software Designs. In *Proc. of ICPP 2005 Workshops*,pp.240-247,June 2005, IEEE Computer Society Press.