Timed Weak Simulation Verification and its application to Stepwise Refinement of Real-Time Software

Satoshi Yamane[†]

[†]*Graduate School of Natural Science&Technology, Kanazawa University, Kakuma-machi, Kanazawa, 920-1192 Japan*

Summary
Real-time software runs over real-time operating systems, and guaranteeing qualities are difficult. In this paper, we propose timed weak simulation relation verification and apply it to a refinement design method of real-time software. Moreover, we apply our proposed method to general real-time software scheduled by fixed-priority preemptive policy.

Key words:
Refinement Design Method, Real-Time Software, Timed Weak Simulation Relation, Verification, Real-Time Scheduling

1. Introduction

Recently almost microprocessors are used in embedded systems. Real-time software runs in embedded systems. As real-time software is reactive and concurrent, and its timing conditions are strict, it is difficult to design real-time software [1]. It is important to specify and verify real-time software [2]. In this paper, we propose timed weak simulation relation verification method based on timed automata [3], and apply it to stepwise refinement of real-time software over fixed priority preemptive schedulers [4].

In general, real-time software is designed by dividing it into tasks [1]. In this case, it is difficult to design real-time software by the following points:

(1)Real-time software consists of many tasks, which concurrently behave. Moreover, tasks interact with external environments. In this situation, it is useful to distinguish between internal events and external events in the sense of process algebra [5].

(2)In real-time software, stepwise refinement is useful [6], and it is important to automatically verify whether the concrete specification refines the abstract specification.

From the above results, we propose the followings:

(1)We use nondeterministic timed automata, which have internal and external events. We construct real-time software by parallel composition of nondeterministic timed automata.

(2)We verify whether the concrete specification refines the abstract specification based on a timed weak simulation.

In general, refinement relations such as language inclusion, timed bisimulation and timed strong simulation are useful.

(1)We can easily and naturally verify fairness and regularity as acceptance conditions by language inclusion. But if we specify verification properties using nondeterministic timed automata, language inclusion problems are undecidable [3]. On the other hand, R. Alur proposed an event-clock automata, which is a determinizable of timed automaton [7,8]. But an event-clock automaton is a subclass of a timed automaton, and accepts a finite timed word (though a general timed automaton accepts an infinite timed word). As the determinization of an event-clock automaton causes an exponential blow-up in the number of locations, the verification cost increases. Moreover, we can not verify some deadlock using language inclusion [9,10].

(2)Timed bisimulation relation is useful for verifying a kind of invariant holding between the more concrete specification and the more abstract specification [11]. On the other hand, timed strong simulation relation is useful for verifying stepwise refinement [12]. But when we stepwise develop specifications, we may add exception procedures to the concrete specification, which are not contained in the abstract specification. Both timed bisimulation relation and timed strong simulation relation are not adequate for this reason.

From the above result, we use timed weak simulation relation in order to verify whether the concrete specification refines the abstract specification.

We survey related works as follows:

(1)In 1992, Cerans has shown that timed strong and weak bisimulation equivalence problem for timed automata are decidable [11]. But he has not developed bisimulation algorithms.

(2)In 1996, Tasiran and his colleagues have developed the verification algorithm of timed strong simulation relation [12]. But they have not developed a timed weak simulation relation.

(3)In 1999, Braberman and his colleagues have

Manuscript received August, 2005.
Manuscript reviced September, 2005.

developed reachability analysis method of preemptive scheduling using timed automata [13]. But they have not developed refinement verification method.

In this paper, we define a timed weak simulation relation, and propose the verification algorithm of timed weak simulation relation. Moreover we apply our proposed method to general real-time software scheduled by fixed-priority preemptive policy. To the best of our knowledge, timed weak simulation verification methods of timed automata have never been developed before now.

The paper is organized as follows: In section 2, we define specification method. In section 3, we define timed weak simulation relation verification method and apply it to stepwise refinement of real-time software. In section 4, we present design support system and some example. Finally, in section 5, we present conclusions.

2. Specification of Real-Time Software

2.1 Syntax and Semantics of Timed Automata

First we define clock and clock interpretation as follows:

Definition 1(*Clock and clock interpretation)*
Given a finite set of variables $X=\{x_1,..,x_n\}$, a valuation is a function $v:X\rightarrow\mathbf{R}$, which assigns a nonnegative real value to each clock variable. We define V^X as the set $[X\rightarrow\mathbf{R}]$. $\mathbf{0}$ denotes the valuation that assigns the value 0 to each $x\in X$. For $\lambda \subseteq X$, $v[\lambda :=0]$ denotes the valuation that assigns the value 0 to each $x\in\lambda$ and agrees with v for all clocks in $X\setminus\lambda$. Moreover, for every $t\in\mathbf{R}$, v+t denotes the clock valuation for which all clocks x take the value v(x)+t. □

Next we define clock constraints.

Definition 2(*Clock constraints)*
For a set X of clock variables, the set Ψ^X of clock constraints ψ is inductively defined by

$$\psi ::= x\sim c \mid \psi_1 \wedge \psi_2,$$

where $\sim\in\{\leqq,=,\geqq\}$, $c\in\mathbf{N}$.
We write $v\models\psi$ if the valuation v satisfies the formula ψ. For each clock $x\in X$, $c_x(\Psi^X)$ denotes the maximal clock constant in Ψ^X. □

Next we define syntax of timed automaton by the followings:

(1)As tasks interact with external environments and other tasks, we distinguish between internal events and external events as shown in Figure 1.

(2)As we think only the external events cause reset actions, internal events can not reset clocks.

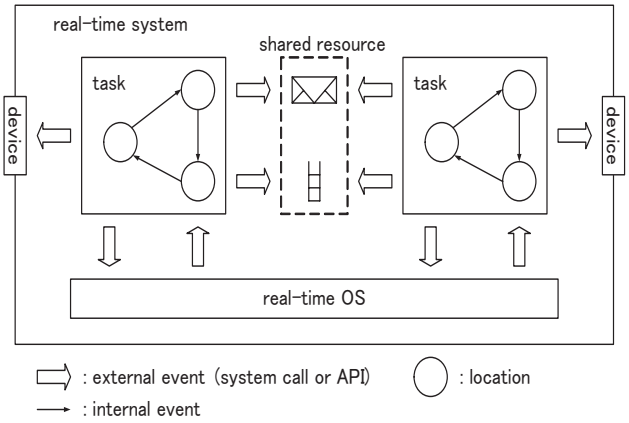

Fig.1 External events and internal events of tasks

Definition 3(*Syntax of timed automat)*
A timed automaton **G** is a tuple $<S,s^{init},\Sigma,X,inv,E>$, where

(1)S is the finite set of locations.

(2)s^{init} is an initial location.

(3)$\Sigma=EXT\cup INT$ is the finite set of events, where EXT is the finite set of external events, INT is the finite set of internal events.

(4)X is the finite set of real-valued variables, called clocks.

(5)$inv:S\rightarrow\Psi^X$ is the invariant function that assigns Ψ^X to each location $s\in S$.

(6)E is the finite set of edges.
Each edge e is a tuple $<s,\sigma,\psi,\lambda,s'>=e\in E$ consisting of the source location s, the target location s', clock constraint $\psi\in\Psi^X$, the set λ of clocks to be reset, $\sigma\in\Sigma$ is an event, where $\lambda=0$ if $\sigma\in INT$. □

Next we formally define semantics of timed automata.

Definition 4.(*Semantics of timed automata)*
A state of **G** is a pair $<s,v>$ containing the location $s\in S$ and the valuation $v\models\psi$. The set of all states is denoted Ω. The initial state is a pair $<s^{init},\mathbf{0}>$. For each state $<s,v>$, the transition is defined as follows:

1. Discrete transitions:

$$\frac{<s,\sigma,\psi,\lambda,s'>\in E,\ v\models\psi,\ v[\lambda:=0]\models inv(s')}{}\ \sigma$$

$<s,v>\rightarrow<s',v'>$

,where $v'= v[\lambda :=0]$.

2. Timed transitions:

$\delta \in \mathbf{R}$, $\forall \delta' \leqq \delta$. $v+\delta' |= inv(s)$ implies $<s,v+\delta>$

$$<s,v> \xrightarrow{\delta} <s',v'>$$

,where $v'=v+\delta$.

A run of timed automaton is an infinite sequence as follows:

$$<s^{init},\mathbf{0}> \xrightarrow{l_1} <s_1,v_1> \xrightarrow{l_2} <s_1,v_1> \xrightarrow{l_3} \cdots\cdots\cdots\cdots$$

,where $<s^{init},\mathbf{0}>$ is an initial state, $<si,v_i> \in \Omega$ is a state, $l_i \in (\sum \cup \mathbf{R})$ is a label. □

In this paper, we assume that timed automaton is nonZeno. It is easy to verify whether a timed automaton is nonZeno or not using HYTECH [14].

2.2 Parallel Composition of Timed Automata

In this paper, we construct real-time software by parallel composition of tasks. We define parallel composition of timed automata as follows:

(1)If the external event of a task is equal to the external event of environments, the task is synchronized with environments by the same external event.

(2)As internal events of tasks are unobservable from environments, internal events of tasks and events of environments are disjoint.

Definition 5(*Parallel composition)*

Let be two timed automata $\mathbf{G_1}=<S_1,s^{init}_1,\sum_1,X_1,inv_1,E_1>$ and $\mathbf{G_2}=<S_2,s^{init}_2,\sum_2,X_2,inv_2,E_2>$. The parallel composition of $\mathbf{G_1}$ and $\mathbf{G_2}$ is the timed automaton $\mathbf{G}=<S,s^{init},\sum,X,inv,E>$, where $\sum_1=EXT_1 \cup INT_1$, $\sum_2=EXT_2 \cup INT_2$. Here $INT_1 \cap \sum_2=\emptyset$ and $\sum_1 \cap INT_2=\emptyset$.

(1)$S \subseteq S_1 \times S_2$

(2)$s^{init}=(s^{init}_1, s^{init}_2)$

(3)$\sum=EXT \cup INT$,

where $EXT=EXT_1 \cup EXT_2$ and $INT=INT_1 \cup INT_2$.

(4)$X=X_1 \cup X_2$

(5)$inv((s_1,s_2))=inv_1(s_1) \wedge inv_2(s_2)$

(6)$<s,\sigma,\psi,\lambda,s'> \in E$,

where for $<s_1,\sigma,\psi_1,\lambda_1,s'_1> \in E_1$ and $<s_2,\sigma,\psi_2,\lambda_2,s'_2> \in E_2$, each element is as follows:

(a)When $\sigma \in \sum_1 \cap \sum_2$, $s=(s_1,s_2)$, $\psi=\psi_1 \wedge \psi_2$, $\lambda=\lambda_1 \cup \lambda_2$, $s'=(s_1',s_2')$.

(b) When $\sigma \in \sum_1$ and $\sum_2$ does not contain σ, $s=(s_1,s_2)$, $\psi=\psi_1$, $\lambda=\lambda_1$, $s'=(s_1',s_2)$.

(c) When $\sigma \in \sum_2$ and $\sum_1$ does not contain σ, $s=(s_1,s_2)$, $\psi=\psi_2$, $\lambda=\lambda_2$, $s'=(s_1,s_2')$. □

2.3 Specification Method

We decide parameters such as priorities and timing constraints by V. Braberman's method [13], which is based on WCRT(Worst Case Response Time) [15].

First we define Worst-Case Response Time as follows:

Definition 6(*Worst-Case Response Time)*

If every task j, j<i, has higher priority than task i, the worst-case response time Ri of task i is given as recursive equation(i=1,…,n). The (k+1)-th worst-case response time $Ri^{(k+1)}$ for task i is as follows($k \geqq 0$):

$$Ri^{(k+1)} = \sum_{j=1}^{i-1} \left(\left\lceil \frac{Ri^{(k)}}{Tj} \right\rceil \right) \times Cj + Ci$$

,where period Ti, execution time Ci, deadline Di of a periodic task i. We can compute $Ri=\lim_{k\to\infty} Ri^{(k)}$ as $Ri^{(0)}=Ci$. $\lceil \rceil$ denotes the integral part. □

Using Ri, we can check whether real-time software is schedulable or not as follows:

> Real-time software is schedulable if the following condition is satisfied:
>
> For $\forall$ i, $Ri \leqq Di$ (i=1,..,n) holds true.[15]

Next we specify real-time software using timed automata. In general, it is not possible to exactly specify preemptive scheduling using timed automata. Therefore, R. Alur and T.A. Henzinger have specified preemptive scheduling using hybrid automata [16]. In this paper, we approximately specify timing constraints by $c_{min} \leqq x \leqq c_{max}$ using timed automata, where we set c_{min} using timing constraints of edges, and set c_{max} using worst-case response time. Therefore, we can realize the automatic verification of timed weak simulation relation. If we specify real-time software of preemptive scheduling using hybrid automata, it is not possible to automatically verify timed weak simulation [17].

Example 1 (*Specification of tasks)*

We specify periodic and sporadic tasks over preemptive schedulers as shown in Figure 2.

(1)Timed automata of tasks contain parameters as follows:
(a)T_1 and T_2 are periodic times,
(b)C_1 and C_2 are minimum response times,
(c)$R_1$and R_2 are worst-case response times,
where $C_1 \leqq R_1 \leqq T_1$ and $C_2 \leqq R_2 \leqq T_2$.
(2)Each node of tasks is as follows:
(a)dormant1 and dormant2 are idle states,
(b)ready1 and ready2 are ready states,
(c)run1 and run2 are execution states.
(3)Each event of tasks is as follows:
(a)start1 and start2 are events, which represent task invocation system calls,
(b)dispatch1 and dispatch2 are events, which represent task dispatch system calls,
(c)end1 and end2 are events, which task terminate system calls.
(4)Each clock of tasks is as follows:
(a)t_1 and t_2 measure response time since task arrivals,
(b)x_1 and x_2 measure execution time. □

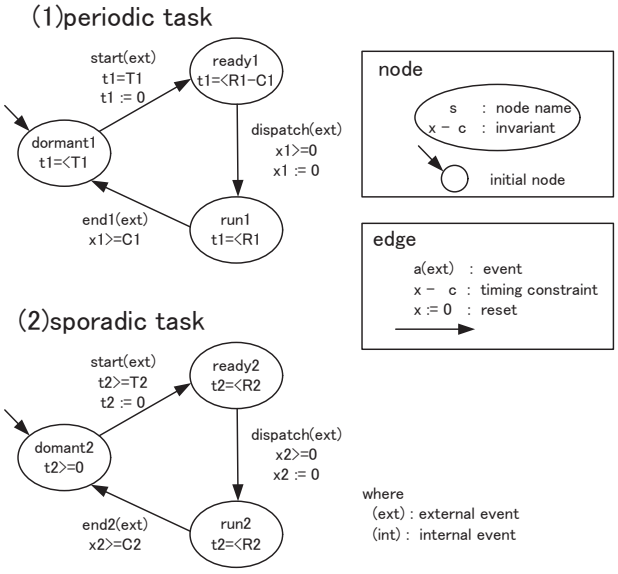

Fig.2 Example of a periodic task and an aperiodic task

3. Refinement Design Method

It is important to design real-time software by stepwise refinement as real-time software is a complex system. In this case, it is important to verify whether the concrete specification is satisfied by the abstract one or not.

First we define a timed weak simulation. Next we define the verification method of a timed weak simulation. Finally we explain the stepwise refinement design method of real-time software.

3.1 A timed weak simulation

First we define observable transitions as follows:

Definition 7(*Observable transitions)*
For each state $<s,v> \in \Omega$ of timed automaton $\mathbf{G}=<S,s^{init}, \sum,X,inv,E>$, observable transitions are defined as follows:

Here Let $\overset{\varepsilon}{\Rightarrow}$ if and only if $(\overset{\tau}{\rightarrow})^*$, where $\sum=EXT \cup INT$ and $\tau \in INT$.
(1)For an external event $\sigma \in EXT$,

Define $<s,v> \overset{\sigma}{\Rightarrow} <s',v'>$ as $<s,v> \overset{\varepsilon}{\Rightarrow} \overset{\sigma}{\rightarrow} \overset{\varepsilon}{\Rightarrow} <s',v'>$.

(2)For delay $\delta_1,..,\delta_k$, $\delta \in \mathbf{R}$,

Define $<s,v> \overset{\delta}{\Rightarrow} <s',v'>$ as $<s,v> \overset{\varepsilon}{\Rightarrow} \overset{\delta 1}{\rightarrow} \ldots \overset{\varepsilon}{\Rightarrow} \overset{\delta k}{\rightarrow} \overset{\varepsilon}{\Rightarrow} <s',v'>$.
In this case, a state $<s,v>$ is called stable from environments, and we denote wait($<s,v>$),
where $\delta_1+..+\delta_k=\delta$. □

Definition 8(*Timed weak simulation)*
Let $\mathbf{G_1}=<S_1,s^{init}_1, \sum_1,X_1,inv_1,E_1>$ and $\mathbf{G_2}=<S_2,s^{init}_2, \sum_2,X_2,inv_2,E_2>$ be two timed automata. A timed weak simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$ is a binary relation Sim $\subseteq \Omega_1 \times \Omega_2$ if the following three conditions are satisfied. Moreover we denote $\mathbf{G_1} \ll \mathbf{G_2}$ if there exists a timed weak simulation relation, where Ω_1 is the set of $<s_1,v_1>$,Ω_2 is the set of $<s_2,v_2>$, $\sum_1=EXT_1 \cup INT_1$, $\sum_2=EXT_2 \cup INT_2$, $s_1,s_1' \in S_1$, $v_1,v_1' \in V^{X1}$, $s_2,s_2' \in S_2$, $v_2,v_2' \in V^{X2}$.
(1)External event condition:
$EXT_1 \subseteq EXT_2$
(2)Simulation condition:
For every $(<s_1,v_1>,<s_2,v_2>) \in$ Sim and for every $\theta \in (EXT_1 \cup \mathbf{R})$,

if $<s_1,v_1> \overset{\theta}{\Rightarrow} <s_1',v_1'>$ then there exists $<s_2',v_2'>$
such that $<s_2,v_2> \overset{\theta}{\Rightarrow} <s_2',v_2'>$ and $(<s_1',v_1'>,<s_2',v_2'>) \in$ Sim.
We show simulation condition in Figure 3.
(3)Initial condition:
$(<s^{init}_1,\mathbf{0}>,<s^{init}_2,\mathbf{0}>) \in$ Sim. □

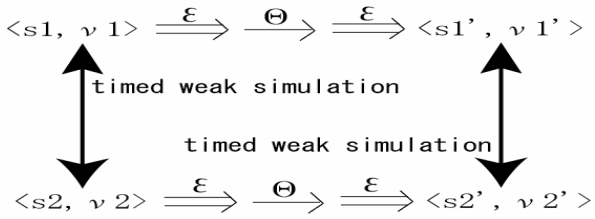

Fig.3 Simulation condition

3.2 Verification method of a timed weak simulation

We achieve a timed weak simulation by converting this check to a finite check on the finitely many equivalence classes, which is called a region weak simulation relation. First we define a region graph, and next define a region weak simulation, and the conversion technique. Finally we show the verification algorithm of a timed weak simulation.

3.2.1. Region graph

Since the number of states is infinite, we cannot possibly build a finite automaton. But if two states with the same location agree on the integral parts of all clock values, and also the ordering of the fractional parts of all clock values, then the runs starting from the two states are very similar. From the above facts, we can construct region graphs, which are finite quotient structures by equivalence relations [3].
First we define equivalence relations of clock values.

Definition 9(*Equivalence relations of clock values)*
Let V^X be the set of clock values, and Ψ^X be the set of clock constraints. For any $t \in \mathbf{R}$, $\lceil t \rfloor$ denotes the integral part of t, and fract(t) denotes the fractional part of t. For v, $v' \in V^X$, v and v' are equivalent iff the following three conditions are satisfied. We denote $v \Leftrightarrow v'$.

(1)For clock $x \in X$, $\lceil v(x) \rfloor$ and $\lceil v'(x) \rfloor$ are the same, or both v(x) and v'(x) are greater than $c_x(\Psi^X)$.
(2)For all $x,y \in X$ with $v(x) \leqq c_x(\Psi^X)$ and $fract(v(x)) \leqq fract(v(y))$ iff $fract(v'(x)) \leqq fract(v'(y))$.
(3)For $x \in X$ with , $v(x) \leqq c_x(\Psi^X)$ and $fract(v(x)) = 0$ iff $fract(v'(x)) = 0$. □

We use [v] to denote the clock region to which v belongs. Next we define the successor of equivalence classes.

Definition 10(*Successor of equivalence classes)*
Let α and β be distinct clock equivalence classes of V^X and Ψ^X. For each $v \in \alpha$ and any $\delta \in \mathbf{R}$, we define the successor of equivalence classes:

(1)We denote $\beta = succ_0(\alpha)$ iff there exists δ such that $\alpha = \beta$ and $\delta \in \beta$.
(2)We denote $\beta = succ_1(\alpha)$ iff there exists $\delta' \leqq \delta$ such that $\alpha \neq \beta$ and $v+\delta' \in \alpha \cup \beta$ and $v+\delta \in \beta$. □

Region is denoted by <s,[v]>, or, <s, α >.

Next we define region graph of timed automaton as follows:

Definition 11(*Region graph)*
For a timed automaton **G**=<S,s^{init}, Σ ,X,inv,E>, the corresponding region graph R(G)=<Q,q^{init},L,N> consists of four tuples:

(1)the finite set of states Q.
(2)the initial state $q^{init} \in Q$, where $q^{init=}<s^{init},\mathbf{0}>$.
(3)the finite set of labels $L=\Sigma \cup SUCC$, where SUCC is the set of labels, which represent successor relations of equivalence classes.
(4)a set of transition relations $N \subseteq Q \times L \times Q$. For any <s, α >, a set of transition relations are defined as follows:

(a) If there exists $<s,v> \xrightarrow{\sigma} <s',v'>$ such that $v' \in \beta$ for each $v \in \alpha$,it is possible to transit to <s', β > by an event,
and we denote $<s, \alpha > \xrightarrow{\sigma} <s', \beta >$.

(b) If there exists $<s,v> \xrightarrow{\delta} <s,v'>$ such that $\beta = succ_1(\alpha)$ and $v' \in \beta$ for each $v \in \alpha$, it is possible to transit to <s, β > by a time delay,
and we denote $<s, \alpha > \xrightarrow{succi} <s, \beta >$ (i=0,1), where $succ_i \in SUCC$. □

Example 2(*Example of region graph)*
We construct region graph from timed automaton **G** in Figure 4. The following transition relations over node s_1 and s_2 are constructed, where initial clock values are $v_1=\mathbf{0}$, $v_2(x)=0 \wedge v_2(y)=1$.

(1)the transition relation over node s_1:
For the state transition

$$<s_1,v_1> \xrightarrow{0.5} <s_1,v_1+0.5> \xrightarrow{0.2} <s_1,v_1+0.7> \xrightarrow{0.3} <s_1,v_1+1> \xrightarrow{a} <s_2,v_2>$$

, the transition relation of regions

$$<s_1,[v_1]> \xrightarrow{succ1} <s_1,[v_1+0.5]> \xrightarrow{succ0} <s_1,[v_1+0.7]> \xrightarrow{succ1} <s_1,[v_1+1]> \xrightarrow{a} <s_2,[v_2]>$$

The equivalence classes of s_1 are classified into $v_1=(x-y=0)$, $[v_1+0.5]=[v_1+0.7]=(0<x=y<1)$, $[v_1+1]=(x=y=1)$.
(2)the transition relation over node s_2:
For the state transition

$\langle s_2,v_2\rangle \xrightarrow{0.2} \langle s_2,v_2+0.2\rangle \xrightarrow{0.3} \langle s_w,v_2+0.5\rangle \xrightarrow{0.5} \langle s_2,v_2+1\rangle \xrightarrow{b} \langle s_1,v_1\rangle$

, the transition of regions

$\langle s_2,[v_2]\rangle \xrightarrow{succ1} \langle s_2,[v_2+0.2]\rangle \xrightarrow{succ0} \langle s_2,[v_2+0.5]\rangle$

$\xrightarrow{succ1} \langle s_2,[v_2+1]\rangle \xrightarrow{b} \langle s_1,[v_1]\rangle$

The equivalence classes of s_2 are classified into $[v_2]=(x=0,y=0)$,$[v_2+0.2]=([v_2+0.5]=(0<x<1<y<2$,$fr(x)=fr(y))$,$[v_2+1]=(x=1,y=2)$.

We can construct equivalence classes (2) and region graph **R(G)** (3) from timed automaton (1). □

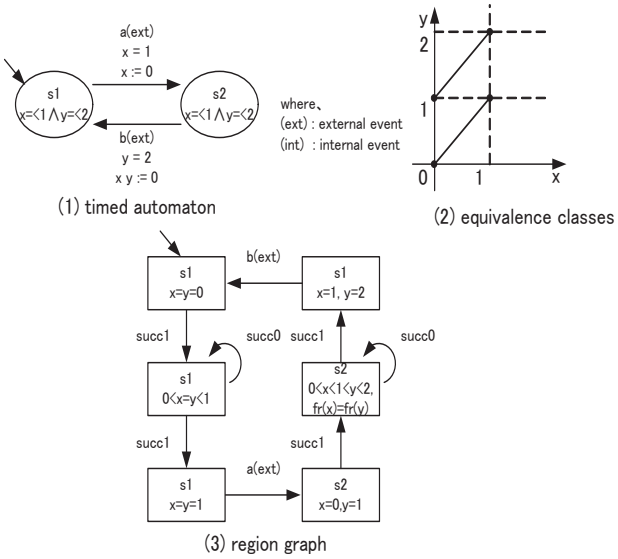

Fig.4. Example of a region graph

3.2.2. Region weak simulation relation

We will show that the problem of checking the existence of a timed weak simulation relation is decidable. We achieve this by converting this check to a finite check on the finitely many equivalence classes of an equivalence relation (what we call region weak simulation relation) defined on parallel composition of timed automata.

We define a region weak simulation relation on parallel composition of timed automata from the following reasons:

(1)We can construct all the pairs of $(\langle s_1,v_1\rangle,\langle s_2,v_2\rangle)\in$ Sim as $\langle(s_1,s_2),v_{12}\rangle$ of parallel compostion of timed automata, where s_1 and s2 are the node and clock value of timed automaton 1, s_2 and v_2 are the node and clock value of timed automaton 2, v_{12} is the clock values of parallel composition of timed automaton 1 and 2. Therefore a set of state pairs Sim, which represent region weak simulation relation, is a subset of a set of states of product automaton (parallel composition of timed automata) $\Omega_{G1 \parallel G2}$. Namely, $Sim \subseteq \Omega_{G1 \parallel G2}$.

(2)As it is easy to trace the relation between v_1 and v_2, we trace it by v_{12} of product automaton.

(3)If one timed automaton has $\langle s_1,v_1\rangle \xrightarrow{\delta} \langle s_1,v_1'\rangle$ and another timed automaton has $\langle s_2,v_2\rangle \xrightarrow{\delta} \langle s_2,v_2'\rangle$, product automaton has $\langle(s_1,s_2,)v_{12}\rangle \xrightarrow{\delta} \langle(s_1,s_2,)v_{12}'\rangle$. Therefore we easily represent two timed automata by product timed automaton.

Definition 12($\mathbf{R(G_1 \parallel G_2)}$)

We construct product timed automaton $\mathbf{G_1 \parallel G_2}$ from $\mathbf{G_1}$ and $\mathbf{G_2}$ by parallel composition, where $EXT_1 \subseteq EXT_2$. We define the region of region graph $\mathbf{R(G_1 \parallel G_2)}$ as $\langle(s_1,s_2),\alpha\rangle$, where $s_1 \in S_1, s_2 \in S_2$, α is the equivalence class of $V^{X1 \cup X2}$ and $\Psi^{X1 \cup X2}$. Let $Q_{G1 \parallel G2}$ be the set of equivalence classes on $\mathbf{G_1 \parallel G_2}$, where $\langle(s_1,s_2),\alpha\rangle \in Q_{G1 \parallel G2}$. With $R(\langle s_1,v_1\rangle,\langle s_2,v_2\rangle)$, denote the equivalence class $\langle(s_1,s_2),\alpha\rangle$ that the state $(\langle s_1,v_1\rangle,\langle s_2,v_2\rangle) \in \Omega_{G1 \parallel G2}$ belongs to. For any region $\langle(s_1,s_2),\alpha\rangle$, observable transitions are as follows:

(1)For an external event $\sigma \in EXT$,

Define $\langle(s_1,s_2),\alpha\rangle \overset{\sigma}{\Rightarrow} \langle(s_1',s_2'),\beta\rangle$

as $\langle(s_1,s_2),\alpha\rangle \overset{\varepsilon}{\Rightarrow} \rightarrow \overset{\sigma}{\Rightarrow} \overset{\varepsilon}{\Rightarrow} \langle(s_1',s_2'),\beta\rangle$.

(2)For $succ_i \in SUCC$,

Define $\langle(s_1,s_2),\alpha\rangle \overset{succi}{\Rightarrow} \langle(s_1',s_2'),\beta\rangle$ (i=0,1)

as $\langle(s_1,s_2),\alpha\rangle \overset{\varepsilon}{\Rightarrow} \rightarrow \overset{succi}{\Rightarrow} \overset{\varepsilon}{\Rightarrow} \langle(s_1',s_2'),\beta\rangle$. □

Next we define a region weak simulation relation on region graph $\mathbf{R(G_1 \parallel G_2)}$

.

Definition 13(*Region weak simulation relation)*

We say that $X \subseteq Q_{G1 \parallel G2}$ is a region weak simulation from $\mathbf{G_1}$ to $\mathbf{G_2}$ iff for each $R(\langle s_1,v_1\rangle,\langle s_2,v_2\rangle) \in X$, the following three conditions are satisfied.

(1)For every $\sigma \in EXT$,

If $\langle s_1,v_1\rangle \overset{\sigma}{\Rightarrow} \langle s_1',v_1'\rangle$, then

$\overset{\sigma}{}$

$<s_2,v_2>\Rightarrow<s_2',v_2'>$

and $R(<s_1,v_1>,<s_2,v_2>)\overset{\sigma}{\Rightarrow}R(<s_1',v_1'>,<s_2',v_2'>)$

such that $R(<s_1',v_1'>,<s_2',v_2'>)\in X$.

(2)If wait$(<s_1,v_1>)$, for any $succ_i\in SUCC$, then

$R(<s_1,v_1>,<s_2,v_2>)\overset{succ}{\Rightarrow}R(<s_1',v_1'>,<s_2',v_2'>)$

such that $R(<s_1',v_1'>,<s_2',v_2'>)\in X$.(i=0,1)

(3) $R(<s^{init}_1, \mathbf{0}>,<s^{init}_2,\mathbf{0}>)\in X$ □

Theorem 1(*Timed weak simulation and region weak simulation)*

For $R(<s_1,v_1>,<s_2,v_2>)\in X$, let $R_X=\{(<s_1,v_1>,<s_2,v_2>)|\ R(<s_1,v_1>,<s_2,v_2>)\in X\}$. R_X is a weak timed simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$ iff X is a region weak simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$.

Proof 1.

We prove it by dividing it into two cases.

(l)To prove that if R_X is a weak timed simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$, X is a region weak simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$:

Assuming that R_X is a weak timed simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$. From the definition, we can directly prove X is a region weak simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$.

(ll)To prove that if X is a region weak simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$, R_X is a weak timed simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$:

Assuming that X is a region weak simulation relation from $\mathbf{G_1}$ to $\mathbf{G_2}$. For some $\theta\in(EXT\cup\mathbf{R})$,$(<s_1,v_1>,<s_2,v_2>)\in R_X$

and $<s_1,v_1>\overset{\theta}{\Rightarrow}<s_1',v_1'>$.

We need to show that

there exists $<s_2',v_2'>$ such that

$<s_2,v_2>\overset{\theta}{\Rightarrow}<s_2',v_2'>$ and $(<s_1',v_1'>,<s_2',v_2'>)\in R_X$.

(1)Whenθ is $\sigma\in EXT$:

From the definition of $\mathbf{R(G_1\parallel G_2)}$, there exists $<s_2',v_2'>$ such that

$<s_2,v_2>\overset{\sigma}{\Rightarrow}<s_2',v_2'>$ and $(<s_1',v_1'>,<s_2',v_2'>)\in R_X$.

(2)When θ is $\delta\in\mathbf{R}$:

Let be $R(<s_1,v_1>,<s_2,v_2>)=<(s_1,s_2),[v]>$ and $R(<s_1',v_1'>,<s_2',v_2'>)=<(s_1',s_2'),[v']>$.

First we define equivalence classes $\alpha_0,\ldots,\alpha_{k+1}$ corresponding to v as follows:

$\alpha_0=[v]$

$\alpha_{i+1}=succ_1(\alpha_i)\quad(0\leqq i\leqq k)$

Next we define real values $\delta_1,\ldots,\delta_k,\delta'$ and clock values $v^{(0)},\ldots,v^{(k)}$ corresponding toα_i as follows:

$v^{(0)}=v$

$v^{(i+1)}=v^{(i)}+\delta_{i+1}\in\alpha_{i+1}\quad(0\leqq i<k)$

$v'=v^{(i)}+\delta'\cup\alpha_{i+1}\quad(i=k)$

As there exists k such that $\delta=\delta_1+\ldots+\delta_k+\delta'$, there are two cases($k\geqq 0$).

(a)k=0:

In this case, as $v'\in\alpha_0\cup\alpha_1$ by$\delta=\delta'$, the transition is as follows:

$<(s_1,s_2),\alpha_0>\overset{succ_i}{\Rightarrow}<(s_1,s_2),\alpha_i>$

Therefore clearly, there exists $<s_2',v_2'>$ such that

$<s_2,v_2>\overset{\delta}{\Rightarrow}<s_2',v_2'>$ and $(<s_1',v_1'>,<s_2',v_2'>)\in R_X$.

(b)k>0:

We consider the following regions as

$<(s_1^{(i)},s_2^{(i)}),v_i>\overset{\delta_{i+1}}{\Rightarrow}<(s_1^{(i+1)},s_2^{(i+1)}),v_{i+1}>$

of $\mathbf{G_1}\parallel\mathbf{G_2}$ $(0\leqq i\leqq k)$.

(i) $<(s_1^{(0)},s_2^{(0)}),\alpha_0>=<(s_1,s_2),[v]>$

(ii) $<(s_1^{(i)},s_2^{(i)}),\alpha_i>\overset{succ_i}{\Rightarrow}<(s_1^{(i+1)},s_2^{(i)+1}),\alpha_{i+1}>$

(iii) $<(s_1^{(i)},s_2^{(i)}),\alpha_i>\in X$

$<(s_1^{(k)},s_2^{(k)}),\alpha_k>\in X$ is inductively derived from $<(s_1,s_2),[v]>$. As $v'\in\alpha_k\cup\alpha_{k+1}$ by $\delta=\delta_1+\ldots+\delta_k+\delta'$, we get the following transition:

$<(s_1^{(k)},s_2^{(k)}),\alpha_k>\overset{succ_j}{\Rightarrow}<(s_1',s_2'),\alpha_{k+j}>$.

Therefore clearly there exists $<s_2',v_2'>$ such that

$<s_2,v_2>\overset{\delta}{\Rightarrow}<s_2',v_2'>$ and $(<s_1',v_1'>,<s_2',v_2'>)\in R_X$. □

3.2.3. Verification algorithm of timed weak simulation

We define the verification algorithm of a timed weak simulation as follows:

Definition 14(*Verification algorithm of a timed weak simulation*)

For the concrete specification $\mathbf{G_L}$ and the abstract specification $\mathbf{G_H}$ we construct $\mathbf{G_L}\parallel\mathbf{G_H}$, and the region graph $\mathbf{R(G_L\parallel G_H)}$. In this case, we define the verification algorithm in order to verify whether there exists a timed weak simulation from $\mathbf{G_L}$ to $\mathbf{G_H}$. Basically first we define $X^{(0)}$, and inductively compute$X^{(k+1)}$ from $X^{(k)}$. First if EXT_2 does not contain EXT_1, there does not exists a timed weak simulation relation. If $EXT_1\subseteq EXT_2$, we compute the followings:

(1)First we compute the relation $X^{(0)}$ =$Q_{GL \parallel GH}$, and initialize k by k:=0.
(2)Next we inductively compute $X^{(k+1)}$ from $X^{(k)}$ by repeating the following procedures (k≧0)
(a) $X^{(k+1)}$=*0*
(b)If R(<s_1,v_1>,<s_2,v_2>)∈$X^{(k)}$, we set
$X^{(k+1)}$= $X^{(k+1)}$∪ {R(<s_1,v_1>,<s_2,v_2>)}
when the two following conditions are satisfied.
(i)For everyσ ∈EXT, if there exists <s_1',v_1'>such that
<s_1,v_1>$\stackrel{\sigma}{\Rightarrow}$<$s_1$',$v_1$'> and <$s_2$,$v_2$>$\stackrel{\sigma}{\Rightarrow}$<$s_2$',$v_2$'>
, R(<s_1,v_1>,<s_2,v_2>)$\stackrel{\sigma}{\Rightarrow}$R(<$s_1$',$v_1$'>,<$s_2$',$v_2$'>),
and R(<s_1',v_1'>,<s_2',v_2'>)∈$X^{(k)}$.
(ii)If wait(<s_1,v_1>), for every succi∈SUCC,
R(<s_1,v_1>,<s_2,v_2>)$\stackrel{succi}{\Rightarrow}$R(<$s_1$',$v_1$'>,<$s_2$',$v_2$'>)
and R(<s_1',v_1'>,<s_2',v_2'>)∈$X^{(k)}$.(i=0,1)
(c)If $X^{(k+1)}$=$X^{(k)}$, go to (3). If $X^{(k+1)} \neq X^{(k)}$, set k:=k+1, and return (2)(a).
(3)If $X^{(k+1)}$ includes R(<s^{init}_1,**0**>,<s^{init}_2,**0**>), we decide X is a timed weak simulation relation. If not so, we decide X is not a timed weak simulation relation. □

As this algorithm can be formalized as the greatest fixpoint computation, the algorithm terminates.

3.3. Stepwise refinement design method

We represent both the abstract specification and the concrete one by nondeterministic timed automata, and verify the consistencies between them by a timed weak simulation relation.

Definition 15(*Stepwise refinement design method)*
The stepwise refinement design method of real-time software consists of the following procedures. We illustrate it in Figure 5.
(1)First we decide task priorities and parameters by WCRT, and specify $\mathbf{TASK_i^{(0)}}$ by timed automata (i=1,…,n), where n is a number of tasks and (0) is the level of refinement.
(2)Next we refine $\mathbf{TASK_i^{(k)}}$ into $\mathbf{TASK_i^{(k+1)}}$, and specify $\mathbf{TASK_i^{(k+1)}}$ by timed automata (k≧0).
(3)Finally, we construct $\mathbf{SOFT^{(k)}}$= $\mathbf{TASK_1^{(k)}}$ ‖ … ‖ $\mathbf{TASK_n^{(k)}}$ and $\mathbf{SOFT^{(k+1)}}$= $\mathbf{TASK_1^{(k+1)}}$ ‖ … ‖ $\mathbf{TASK_n^{(k+1)}}$. We revise $\mathbf{SOFT^{(k+1)}}$ until there exists a timed weak simulation relation from $\mathbf{SOFT^{(k+1)}}$ to $\mathbf{SOFT^{(k)}}$.
(4)We repeat the above step (1)-(3), and specify the final one. □

4. Example of refinement design

In this paper, we show our proposed method effective by a plant system.

4.1. Design support system

In this paper, we implement our proposed method as design support system as shown in Figure 6. The design support system is implemented by C++ language(7000 lines) on Sun Blade1000(CPU UltraSPARC-III 900MHz, Main memory 1GB).

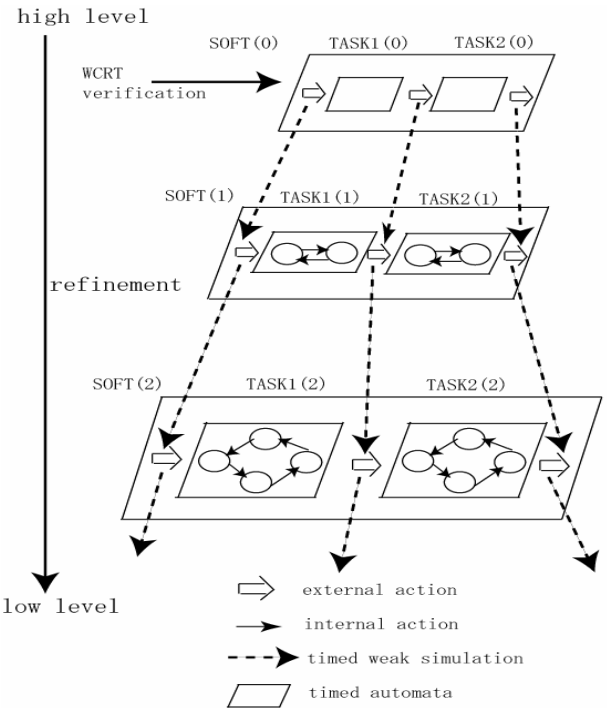

Fig.5 Image of a hierarchical refinement design method

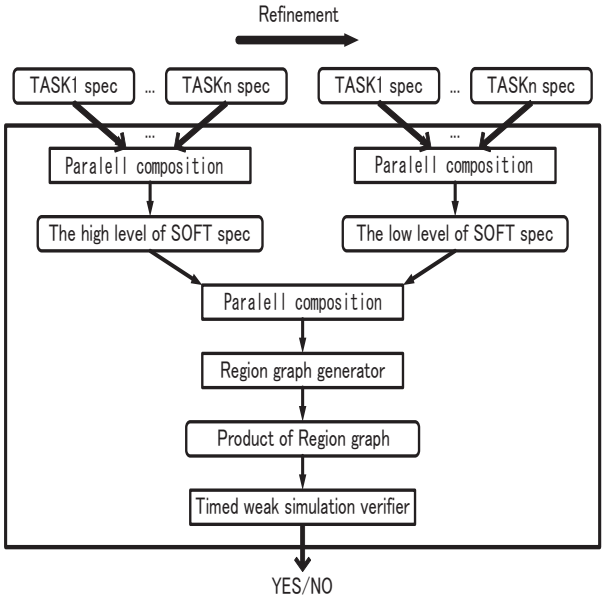

Fig.6 Configuration of the design support system

4.2. A plant system

The plant system is an embedded real-time system, and behaves by fixed priority preemptive scheduling.

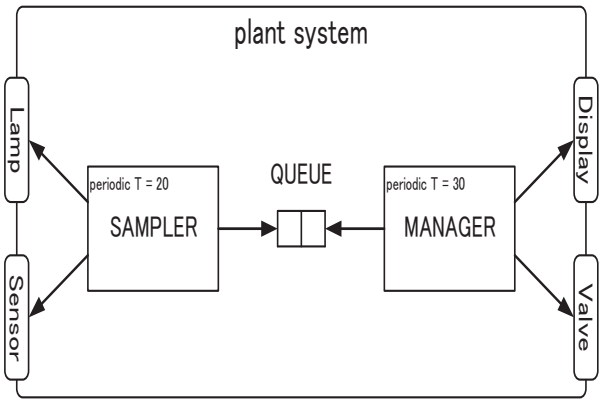

Fig.7 Architecture of the plant system

The plant system is shown in Figure 7. The plant system samples data from Sensor, and controls Valve, and outputs information on Display. If Sensor is in bad condition, this information is announced to administrator by Lamp. Even if Sensor is in bad condition, the plant system continue to behave by control data. The real-time software of the plant system consists of two periodic tasks such as SAMPLER and MANAGER. The two tasks communicate with each other by QUEUE.

We explain each task and resource as follows:

(1)A periodic task SAMPLER:

We design SAMPLER as T_1=20 and C_1=6. SAMPLER samples signals from Sensor, and decides the state of the plant by analyzing signals. If the state changes, SAMPLER sends control data to MANAGER via QUEUE (This processing takes at least 5 time). If the state does not change, SAMPLER does not send control data (This processing takes at least 3 time).

(2)A periodic task MANAGER:

We design MANAGER as T_2=20 and C_2=10. MANAGER receives control data from SAMPLER via QUEUE. If there exists data in QUEUE, MANAGER analyzes the data, and controls Valve, and indicates system information on Dispaly (This processing takes at least 8). If there does not exist data, MANAGER updates system information on Display(This processing takes at least 4).

(3)Shared resource QUEUE:

QUEUE can store two data, and is used for task commmunications. The access to QUEUE takes 1.

In this paper, we think the plant system is implemented by rate monotonic scheduling. Therefore this rate monotonic scheduling algorithm assigns priorities to tasks based on their periods: the shorter the period, the higher the priority. Moreover, we assume Di=Ti (i=1,2). We compute WCRT(Worst Case Response Time) [15]. From the results, we can determine two tasks are schedulable. The parameters are shown in Table 1.

Table 1 List of timed parameters of tasks

task	Ti	Ci	Bi	Ri
SAMPLER	20	6	1	7
MANAGER	30	10	0	16

4.2.1. Specification of the plant system

We design the plant system, and specify it by timed automata.

First we show external events in Table 2. External events are classified into three types such as system call, task control and API(APplication Interface).

Table 2 List of external task events

TYPE	**EXTERNAL EVENT**	**EXPLANATION**
system call	starti	task start
system call	endi	task terminate
system call	push_queue	send to QUEUE
system call	pop_queue	Receive from QUEUE
task control	dispatchi	CPU assignment

API	read_sensor	Sensor input
API	write_lamp	Lamp output
API	write_display	Display output
API	write_valve	Valve output

Next we show the shared resource QUEUE in Figure 8.

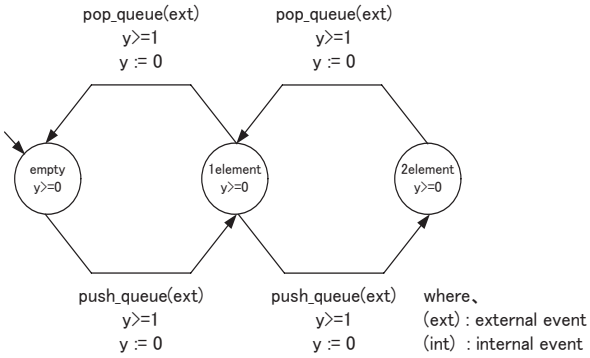

Fig. 8 Specification of the shared resource QUEUE

Next we design the abstract specifications **SAMPLER**$^{(0)}$ and **MANAGER**$^{(0)}$, and specify them by timed automata, and show them in Figure 9 and 10. In abstract specification, we specify only external events and abstract behaviors. For example, we specify **SAMPLER**$^{(0)}$ by nondeterministically behaving write_lamp.

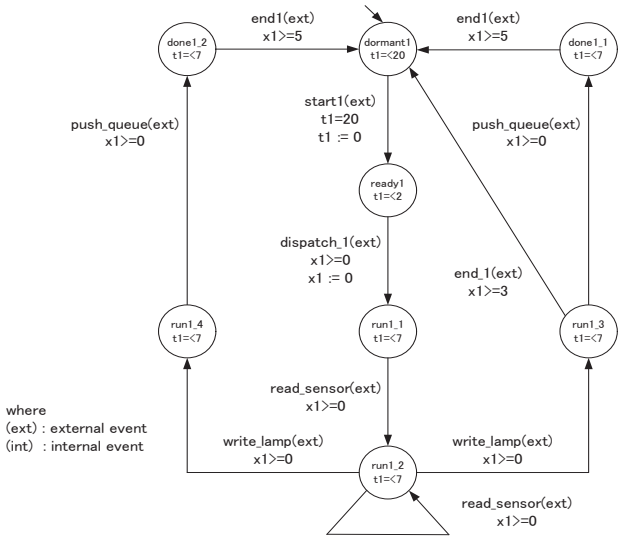

Fig. 9. Specification of the periodic task **SAMPLER**$^{(0)}$

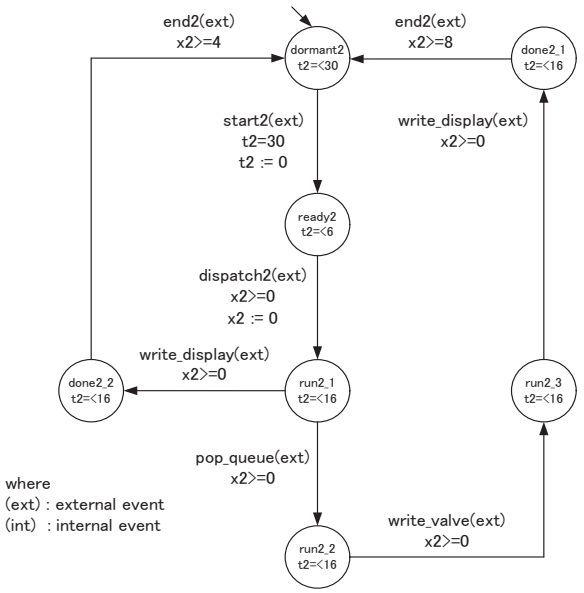

Fig. 10. Specification of the periodic task **MANAGER**$^{(0)}$

Finally we design the concrete specifications **SAMPLER**$^{(1)}$ and **MANAGER**$^{(1)}$, and show them in Figure 11 and 12. We refine the abstract specification into the concrete one by adding internal behaviors, transforming nondeterministic behaviors into deterministic behaviors and specifying detailed timing constraints. For example, **SAMPLER**$^{(1)}$ executes read_sensor once or twice, and if it fails, **SAMPLER**$^{(1)}$ executes make_alternate_data. As the deviation of control data is larger the processing of write_valve takes longer time, in **MANAGER**$^{(1)}$, derivation_Small takes 1, derivation_Medium takes 2 and derivation_Large takes 3.

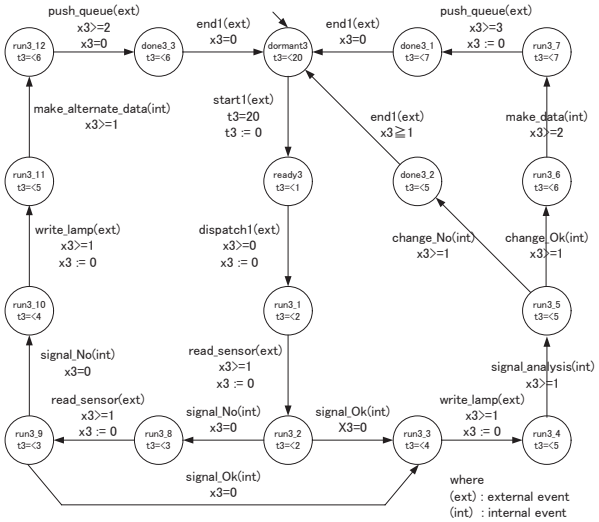

Fig. 11. Specification of the periodic task **SAMPLER**$^{(1)}$

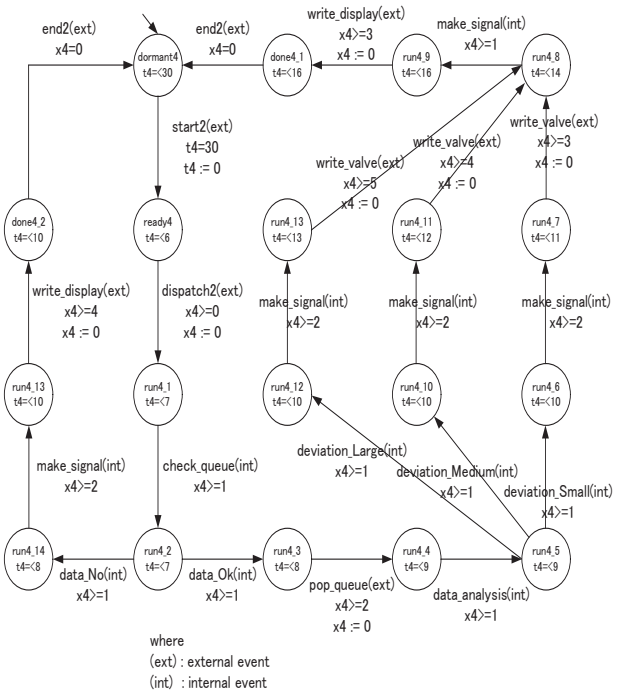

Fig. 12. Specification of the periodic task MANAGER$^{(1)}$

4.2.2. Verification experiment

We verify whether **SOFT$^{(1)}$** ≪ **SOFT$^{(0)}$** holds true or not using our design support system, where the abstract specification is defined as **SOFT$^{(0)}$=SAMPLER$^{(0)}$ ‖ MANAGER$^{(0)}$ ‖ QUEUE**, the concrete specification is defined as **SOFT$^{(1)}$=SAMPLER$^{(1)}$ ‖ MANAGER$^{(1)}$ ‖ QUEUE**. We measured required memory and execution time using ps command, and show them in Table 3. In region graph **R(SOFT$^{(0)}$ ‖ SOFT$^{(1)}$)**, the number of the equivalence classes is 62367, the number of regions of $\mathbf{Q}_{\mathbf{SOFT(0)\,\|\,SOFT(1)}}$ is 1645498 and the number of members of region weak simulation X is 1408290.

Table 3. List of results of experiments

the num. of regions	the num. of transitions	the num. of clocks	required memory	execution time
1645498	3406342	10	325744 kB	62.30min

The members of the set X, which are contained in the region weak simulation relation are the followings:

An initial state pair
R(<(dormant$_1$,dormant$_2$,empty),[t_1=x_1=t_2=x_2=y=0]>,
< (dormant$_3$,dormant$_4$,empty),[t_3=x_3=t_4=x_4=y=0] >),
…….
…….
R(<(run$_{1\text{-}3}$,run$_{2\text{-}2}$,empty),[$t_1 \leqq 7, x_1 \geqq 0, t_2 \leqq 16, x_2 \geqq 0, y \geqq 0$]>,< (run$_{3\text{-}4}$,run$_{4\text{-}4}$,empty),[$t_3 \leqq 5, x_3 \geqq 0,\ 2 \leqq t_4 \leqq 9, x_4 \geqq 1, y \geqq 0$] >),
R(<(done$_{1\text{-}1}$,run$_{2\text{-}2}$,1element),[$t_1 \leqq 7, x_1 \geqq 0, t_2 \leqq 16, x_2 \geqq 0, y \geqq 0$]>,<(done$_{3\text{-}1}$,run$_{4\text{-}4}$,1element),[$t_3 \leqq 7, x_3 \geqq 0,\ 2 \leqq t_4 \leqq 9, x_4 \geqq 1, y \geqq 0$] >),
…….
…….

For example, we focus on
R(<(run$_{1\text{-}3}$,run$_{2\text{-}2}$,empty),[$t_1 \leqq 7, x_1 \geqq 0, t_2 \leqq 16, x_2 \geqq 0, y \geqq 0$]>,<(run$_{3\text{-}4}$,run$_{4\text{-}4}$,empty),[$t_3 \leqq 5, x_3 \geqq 0,\ 2 \leqq t_4 \leqq 9, x_4 \geqq 1, y \geqq 0$]>).
Here we consider the followings:

$$\langle run_{1\text{-}3},[t_1 \leqq 7, x_1 \geqq 0]\rangle \overset{\varepsilon\ \ push\_queue\ \varepsilon}{\Longrightarrow} \rightarrow \Rightarrow \langle done_{1\text{-}1},[t_1 \leqq 7, x_1 \geqq 0]\rangle,$$

$$\langle run_{3\text{-}4},[t_3 \leqq 5, x_3 \geqq 0]\rangle \overset{\varepsilon\ \ push\_queue\ \varepsilon}{\Longrightarrow} \rightarrow \Rightarrow \langle done_{3\text{-}1},[t_3 \leqq 7, x_3 \geqq 0]\rangle$$

$$\langle empty,[y \geqq 0]\rangle \overset{\varepsilon\ \ push\_queue\ \varepsilon}{\Longrightarrow} \rightarrow \Rightarrow \langle 1element,[y \geqq 0]\rangle.$$

Therefore, we can get the followings:
R(<(done$_{1\text{-}1}$,run$_{2\text{-}2}$,1element),[$t_1 \leqq 7, x_1 \geqq 0, t_2 \leqq 16, x_2 \geqq 0, y \geqq 0$]>,<(done$_{3\text{-}1}$,run$_{4\text{-}4}$,1element),[$t_3 \leqq 7, x_3 \geqq 0,\ 2 \leqq t_4 \leqq 9, x_4 \geqq 1, y \geqq 0$] >).

As an initial region is contained in X , **SOFT$^{(1)}$** ≪ **SOFT$^{(0)}$** is satisfied. Namely, there exists a timed weak simulation from the concrete specification to the abstract one.

Though the verification cost is large, we can specify both the abstract and the concrete specifications of real-time software using timed automata, and automatically verify whether there exists a timed weak simulation from the concrete specification to the abstract one.

5. Conclusion

In this paper, we have proposed the following stepwise refinement design methodology of real-time software.

(1)First as we extend general timed automata by distinguishing between internal and external events, this timed automata are suitable for stepwise specifying and designing real-time software.

(2)Next we automatically verify whether there exists a timed weak simulation relation from the concrete specification to the abstract one.

To the best of our knowledge, a timed weak simulation verification methods of timed automata have never been developed before now.

We are now planning to do the following works:

(1)We will develop the effective verification techniques such as Assume-Gurantee and Abstraction in order to verify large software.

(2)We will apply our proposed method to practical problems.

References

[1]J.W.S. Liu. : Real-Time System, Prentice-Hall (2000).

[2]K.M. Kavi. : Real-Time Systems, Abstractions, Languages, and Design Methodologies, IEEE Computer Society (1992).

[3]R. Alur, D.L. Dill. : A theory of timed automata, Theoretical Computer Science, Vol. 126, pp.183-235 (1994).

[4]Giorgio C. Buttazzo. : Hard Real-Time Computing Systems, Kluwer Academic Publishers (1987).

[5]R. Milner. : Communication and Concurrency, p.260, Prentce Hall (1989).

[6]R.H. Thayer, M. Dorfman. : System and Software Refinements Engineering, IEEE Computer Society (1990).

[7]R. Alur, L. Fix, T.A. Henzinger. : Event-Clock Automata: A Determinizable Class of Timed Automata, in Proc. of CAV'94, LNCS 818, pp.1-13 (1994).

[8]C. Dima. : Removing slient transitions from event-clock automata, in Proc. of CITTI 2000, pp 75-81 (2000).

[9]R. Milner. : Operational and Algebraic Semantics of Concurrent Processes. Handbook of Theoretical Computer Science, Volume B: Formal Models and Sematics, J. van Leeuwen, ed., North-Holland Pub. Co./MIT Press, pp. 1201-1242 (1990).

[10]A. Pnueli. : Linear and Branching Structures in the Semantics and Logics of Reactive Systems. ICALP 1985, volume 194 of LNCS, pp. 15-32 (1985).

[11]K. Cerans. : Decidability of bisimulation equivalences for parallel timer processes. LNCS 663, pp.269-300, Springer Verlag (1992).

[12]S.Tasiran, R.Alur, R.P.Kurshan, and R.K.Brayton. : Verifying Abstractions of Timed Systems. LNCS 1119, pp.546-562, Springer Verlag (1996).

[13]V.Braberman, M.Felder. : Verification of Real-Time Designs: Combining Scheduling Theory with Automatic Formal Verification, LNCS 1687, pp.494-510(1999)

[14] T. Henzinger, X. Nicollin, J. Sifakis, S. Yovine. : Symbolic model checking for real-time systems, Information and Computation 111, pp.193–244(1994).

[15]M.Joseph. : Real-Time System Specification, Verification and Analysis. Prentice Hall (1996).

[16]R. Alur, T.A. Henzinger, P.-H. Ho. : Automatic symbolic verification of embedded systems. IEEE Trans. on Software Engineering 22(3), pp.181-201(1996).

[17]T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. : What's decidable about hybrid automata? Journal of Computer and System Sciences 57, pp.94–124(1998).

Satoshi Yamane received the Ph.D in Computer Science from Kyoto University in 1997. He was an Associate Professor at Kagoshima University in 1999. He jointed Kanazawa University , and is currently a Professor. His research interests include formal verification of real-time and hybrid systems. He is a member of ACM, IEEE, and EATCS.